Disk Storage and File Systems with Quality-of-Service Guarantees

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Diplom-Informatiker Lars Reuther
geboren am 25. November 1973 in Dresden

Betreuender Hochschullehrer: Prof. Dr. rer. nat. Hermann Härtig
Technische Universität Dresden

Gutachter: Prof. Dr. rer. nat. Hermann Härtig
Technische Universität Dresden

Prof. Dr.-Ing. Wolfgang Lehner
Technische Universität Dresden

Prof. Scott A. Brandt
University of California, Santa Cruz, USA

Tag der Verteidigung: 18. Mai 2006

Dresden im Mai 2006

Disk Storage and File Systems with Quality-of-Service Guarantees

Lars Reuther
Technische Universität Dresden

May 2006

Abstract

Modern disk-storage systems have to accomplish the requirements of a variety of application classes. Applications that process continuous-media data such as video and audio streams require the storage system to guarantee sustained bandwidths. Interactive applications demand the storage system to ensure bounded response times, posing timing constraints on the execution of individual disk requests. Traditional timesharing applications may require both high throughput or overall short response times. With the described applications being more and more used together in todays computing systems, the disk-storage subsystems have to efficiently combine the different requirements of this application mix.

In this thesis, I develop the design of a storage system that comprehensively addresses the various challenges posed by including the support for quality-of-service guarantees in disk-storage systems. The presented storage system provides three main properties. First, the admission control includes the support for statistical guarantees to increase the share of the disk bandwidth that can be utilized by the admission control. Second, the disk-request scheduling clearly separates the enforcement of real-time guarantees from the task to establish the optimal execution order of the requests, and it provides a flexible mechanism to combine the execution of requests with different quality-of-service requirements. Finally, the file system addresses both the needs of the former two elements of the storage system and of the various file types used by the applications by providing a flexible block-allocation policy and customized client interfaces. I show the implementation of the presented designs with the DROPS Disk-Storage System and I provide a detailed evaluation based on this implementation.

Acknowledgements

First and foremost, I would like to thank all the people who helped making my time at the TU Dresden a lasting experience. I had the privilege to meet a variety of interesting people who not only influenced my work but also had an influence on my own personality.

I would like to thank my supervisor, Prof. Hermann Härtig, for his support and for providing an open work environment at the Operating Systems Chair. Furthermore, I would like to thank the reviewers of my thesis, Prof. Wolfgang Lehner and Prof. Scott Brandt, for their helpful comments and quick responses.

Many people helped making this thesis to become reality. I am thankful to the members of the Operating Systems Group at the TU Dresden for the team work of the last years. In particular, I want to thank Dr. Claude-Joachim Hamann for the many insightful discussions and his patience in getting all the math right. Martin Pohlack helped a lot to move this work in the right direction and to make the system work.

I want to specifically thank Volkmar Uhlig, who is a good friend and a source of never-ending discussions. His constant pushing helped a lot maintaining the motivation to accomplish this thesis and to go on with my personal career.

Last but not least, I want to thank my parents for their support and patience. And I want to thank the members of the Club Dürerstraße for the countless hours of fun we had over the last years.

Table of Contents

Chapter 1

Introduction

Incorporating the support for quality-of-service guarantees into disk-storage systems has drawn attention for a quite while now. The most prominent reason for this interest are multimedia systems that include the processing of video and audio streams. The processing of such *continuous-media* data poses requirements on both the timely handling of the data and the bandwidth provided by the storage system. But these continuous-media applications are not the only cause for the incorporation of quality-of-service guarantees, other applications such as real-time databases or interactive media applications also require the timely handling of their storage requests. Resulting from a more recent trend, these multimedia applications have become a more and more significant part of the commodity use of computer systems.

To incorporate the support for quality-of-service guarantees, the disk-storage system essentially needs to provide two properties. First, an *admission control* is required to decide whether the system is able to handle an additional load, based on the current utilization of the system. Second, the storage system must deploy a scheduling policy that ensures that the guarantees given to its clients are met, once they are accepted by the admission control. The enforcement of the guarantees particularly requires the scheduling policy to execute the disk requests in an order such that the execution of the requests is finished prior to the deadline set by the client. The admission control can provide different types of guarantees. With *hard real-time guarantees*, the storage system ensures that all storage requests of a client are successfully executed in time, which requires the admission control to be based on worst-case assumptions. In contrast, with *soft* or *statistical real-time guarantees* not all requests are necessarily executed, but the system ensures an adequately high percentage of the requests to be executed, which relaxes the assumptions used by the admission control.

Statistical guarantees are of particular interest with disk-storage systems for two reasons. First of all, because of the physical design of disk drives the execution of disk requests shows a poor ratio of the average-case execution time and the worst-case execution time, the latter can exceed the average-case execution time by an order of magnitude. An admission control based on such worst-case assumptions results in a low disk utilization. The utilization can be significantly improved by using a statistical admission model based on a description of the real behavior of the disk, for instance using random variables to describe the request execution time, and tolerating a small probability that the execution of a disk requests misses its deadline. Second, especially multimedia applications are able to cope with the occasional miss of a deadline or the loss of a data packet, but cannot afford the low utilization caused by the hard real-time guarantees. For instance, video-playback applications are able to resynchronize to the video stream after a lost packet, and the resulting video quality remains acceptable as long as the loss rate stays within reasonable boundaries.

Resulting from multimedia applications becoming part of the commodity usage, the focus to incorporate the enforcement of quality-of-service guarantees shifts to include this enforcement with commodity computing systems, and away from dedicated systems such as continuous-media servers. Incorporating quality-of-service guarantees with traditional, best-effort disk-storage systems in a *mixed-media* storage system creates additional challenges, as the storage system needs to combine the requirements of the traditional system, such as short response times, high throughput, or the fairness between the clients of the storage system, with the enforcement of the quality-of-service guarantees. In particular, the enforcement of these guarantees can be contrary to the requirements of best-effort systems, as the storage system might be forced to prioritize the execution of a disk request to meet a guarantee, resulting in either delaying the execution of another request or an execution order of the disk requests not achieving the best possible throughput. On the other hand, the workload created by the traditional systems is less predictable than the load of continuous-media applications, which has to be considered by the admission control.

To summarize, the efficient support for quality-of-service and real-time guarantees in disk-storage systems requires first an admission control that incorporates statistical guarantees to improve the utilization of the system as well as is able to deal with unpredictable workloads such as generated by best-effort applications; and second a disk request scheduling policy that both enforces the guarantees given by the admission control and that still provides an adequate performance to the overall system. An extensive amount of work exists addressing individual elements of these requirements, but there is no work to my knowledge that combines theses requirements within a single disk-storage system.

A variety of approaches exists to provide quality-of-service guarantees with dedicated continuous-media servers, such as video-on-demand systems. The execution of the disk requests is typically organized in rounds, assigning the requests of each client a fixed position within each round. This organization provides only limited flexibility to incorporate other requests than those of the continuous-media applications into the schedule. The existing solutions to incorporate statistical guarantees with continuous-media servers are based on a common approach, the admission control calculates the probability that requests miss their deadlines using stochastic models of the behavior of the disk, and accepts the workload as long as that probability does not exceed a predefined value. The models used by the admission control to calculate these probabilities require the knowledge of the entire workload handled by the disk, which does not allow to incorporate unpredictable workloads, such as requests generated by best-effort applications. The available approaches to provide mixed-media systems focus on the disk-request scheduling combining the execution of requests with different quality-of-service guarantees, but lack comprehensive admission control models.

In this thesis, I develop the design of a disk-storage system that overcomes the described limitations. The design addresses the needs of advanced storage systems that have to combine the requirements of a variety of applications classes, ranging from multimedia applications to traditional timesharing applications. I apply the developed designs to the DROPS *Disk-Storage System*, it provides the following properties:

- The admission control is able to provide both hard real-time guarantees and statistical real-time guarantees. With statistical guarantees, the admission control ensures that a requested percentage of disk requests of a client is executed successfully. In contrast to the existing approaches to support statistical guarantees, the admission control calculates the amount of time required by the execution of the disk requests to achieve the requested percentage, and clients are only admitted if this time demand can be met by the system. The calculation of the time demand is based on a comprehensive probabilistic model that uses random variables to describe the execution times of disk requests. The knowledge of this time demand provides the request scheduling with the required information to be able to enforce these guarantees also with the presence of other clients, such as best-effort applications.

- The disk-request scheduling policy uses the times calculated by the admission control to reserve an appropriate share of the available time to execute the requests of those real-time clients. The enforcement of these reservations is done using a flexible approach, exploiting the *slack time* provided by the requests of real-time clients to both include requests of best-effort applications with the scheduling decisions as often as possible, and to maximize the disk throughput. Upon each scheduling decision, the request scheduler calculates the subset of the outstanding disk requests that can be included in the current scheduling decision without violating any of the real-time guarantees. The scheduler then picks the request that is executed based on the rotational positions of the requests to achieve a good overall disk utilization.
- Finally, the design of the file system accomplishes the requirements of both the admission model and the disk request scheduling. In particular, the file system ensures that real-time requests can by executed using an adequate request size by providing an allocation policy that supports various block sizes; and it uses a streaming client interface that matches the stream model used by the admission control.

The presented work concentrates on providing the described properties with a single disk. The work can be applied to systems consisting of multiple disks using established approaches such as coarse-grained striping, deriving the requirements posed on each individual disk by breaking down the overall requirements of the clients to each disk.

Organization of the Thesis

This thesis is organized as follows. The next chapter gives an overview of the related work and introduces the *Dresden Real-Time Operating System* (DROPS) that provides the environment for this thesis. In particular, Section 2.3.1 discusses the *Quality-Assuring Scheduling* (QAS), which forms the basis for the admission model developed in this thesis.

Chapter 3 describes the support for quality-of-service guarantees for the disk storage. Section 3.1 introduces the resource model used by the admission control, which is described in Section 3.2. Section 3.3 discusses the request-scheduling policy used to enforce the quality-of-service guarantees.

Starting from the requirements posed by the disk-scheduling model, Chapter 4 introduces the design of the DROPS File System, which includes the block-allocation policy described in Section 4.1, a description of the structure of the file-system metadata in Section 4.2 and a discussion of the client interface and the resulting execution model used by the file system in Section 4.3.

Chapter 5 outlines the implementation of the presented designs with the DROPS *Disk-Storage System*. Section 5.2 discuses the implementation of the disk request scheduling and Section 5.3 gives an overview of the implementation of the file system, particularly describing the streaming interface.

Based on the implementation with the DROPS Disk-Storage System, Chapter 6 provides an evaluation of the presented designs. The evaluation includes an analysis of the admission model supporting the statistic quality-of-service guarantees in Section 6.2, the examination of the request-scheduling policy in Section 6.3, the discussion of the integration of the storage system into the overall DROPS architecture in Section 6.4, and finally an evaluation of the file-system design in Section 6.5.

Finally, Chapter 7 concludes the thesis with a summary and suggestions for future work.

Chapter 2

Foundations and Related Work

In this chapter, I will present the basics of my work, define the scope of this thesis and discuss related approaches to incorporate quality-of-service and real-time guarantees into disk-storage systems.

The literature offers an extensive body of work aiming to provide quality-of-service and real-time guarantees with disk-storage systems. Based on the motivation presented in the introduction, the discussion of the related work will concentrate on approaches that aim to combine the diverse requirements of *mixed-media systems* and on systems that deploy statistical admission models. This chapter also includes a description of the *Dresden Real-Time Operating System* (DROPS) [37]. With the *Quality-Assuring Scheduling* (QAS) [31], DROPS provides the theoretical foundation of the scheduling models developed in this thesis, and the environment offered by DROPS is used to evaluate these models.

The chapter is organized as follows. First, Section 2.1 provides an overview of the foundations and the terminology used in this thesis. Section 2.2 discusses related approaches to incorporate quality-of-service and real-time guarantees with disk-storage systems. Finally, Section 2.3 describes DROPS, particularly focusing on the concepts of the Quality-Assuring Scheduling.

2.1 Foundations

This section describes the foundations and introduces the terminology used throughout this thesis. In particular, the section provides an overview of the general structure of disk-storage systems, including a discussion of the challenges posed on the disk-request scheduling; and it further describes the requirements posed on a disk-storage system by multimedia and continuous-media systems.

2.1.1 Disk-Storage Systems

Disk drives are used in a variety of different environments to store data, ranging from hand-held audio players to large-scale storage servers consisting of networked storage nodes. In spite of the diverse structure of these systems, they all have to accomplish two common requirements [26, 85]:

1. The storage system has to provide mechanisms to organize the data stored on the disks. It includes policies to manage the space provided by the disk, mechanisms to keep track of which data on the disk belong to the files of the clients of the storage system, and mechanisms that

allow clients to access the data stored on the disk. This functionality is typically combined into a *File System*.

2. Disk drives can only execute one request at a time, requiring the storage system to arrange the order of the execution of requests concurrently arriving at the storage system. The execution order of disk requests can have a large effect on the performance achieved by the disk, resulting from the delays caused by the mechanical design of the disk. The aims and challenges of the *Disk-Request Scheduling* are discussed in detail in the next section.

The designs of both the file system and the disk-request scheduling depend on the requirements posed by the individual disk-storage system, and each storage system might have to provide additional properties. Especially, networked storage servers require appropriate network protocols to connect the storage nodes, but such properties are beyond the scope of this thesis.

2.1.2 Disk-Request Scheduling

Determining the execution order of the requests handled by the disk is a crucial task of any disk-storage system. The execution order has a large influence on both the *bandwidth* that can be achieved by the disk (i.e., the amount of data that can be read or written by the disk within a time interval) and the *response time* observed by the clients of the storage system (the response time consists of the time a requests is queued within the storage system and the time required to execute the request by the disk).

A major cause of this influence is the mechanical design of the disk, outlined in Figure 2.1(a). Disk drives store data on one or several rotating platters, using a *read/write head* to access the data on the magnetic surface of the platters. The data is organized in concentric *tracks*, requiring the disk head to be moved between the various tracks of the surface to access the data. Within a track, the data is organized in *sectors* that form the smallest unit that can be accessed by the disk, the typical size of a sector is 512 Byte. Modern disks exploit the different lengths of the concentric tracks by storing more sectors in the longer outer tracks of the disk than in the shorter inner tracks, dividing the disk in various *zones* where each zone stores a different number of sectors within its tracks. The current interfaces used between the disk drive and the host system (such as IDE or SCSI [72]) mostly use *logical* block addresses to specify the target sectors of a request, which must be translated into the actual position of the sectors on the disk platters. The most common mapping is a linear mapping between the logical addresses and the sector positions, starting with the numbering at the outermost tracks of all surfaces (i.e., the outermost *cylinder*) and first enumerating all sectors of this cylinder before switching to the second cylinder, and so on.

Figure 2.1(b) outlines the principle procedure of the execution of a request by the disk. The overall time required to process a disk request can be subdivided into three main elements. First, the disk head must be moved to the track containing the target sectors (*seek*), the required time depends on the distance between the starting position of the disk head and the target track. Second, the disk head needs to wait until the target sectors arrive at the disk head (*rotational delay*). This delay depends on the angle distance between the point where the disk head arrives at the track and the position of the target sectors, and the rotational speed of the disk. Finally, the disk head reads the data from the disk or writes the data to the disk. In addition to these three main elements, the overall execution time includes more delays such as the time required to process the requests within the disk controller. These times are usually combined in a fixed *command overhead*.

The disk-request scheduler must consider this execution procedure in order to determine an optimal execution order of the requests. Only the final of the main three described stages of the request

(a) Mechanical Structure of a Disk Drive.

(b) Request Execution.

Figure 2.1: Mechanical structure of a disk drive and procedure of the request execution. Figure (a) illustrates the mechanical design of a disk drive. Figure (b) outlines the procedure of the execution of a request by the disk (omitting the command processing within the disk controller and the data transfer between the disk and the host system).

execution transfers data between the disk and the host system, and therefore contributes to the utilization of the disk drive. Thus, the utilization of the disk, and the bandwidth resulting from this utilization, depends on the ratio between the time required to position the disk head (consisting of the seek and the rotational delay) and the time actually spent transferring data between the disk and the host system. This behavior defines one of the most important goals for the disk-request scheduling, which is to reduce the seek time and the rotational delay to improve the disk utilization. To achieve this goal, several policies exist to order the requests available to the disk-storage system and to pick the next request that is executed by the disk. The *Shortest Seek Time First* (SSTF) [74] scheduling policy always selects the request that results in the shortest track distance between the current position of the disk head and the next request. The SCAN policy [74] (also known as *elevator* policy) is similar to SSTF, but moving the disk head only in a single direction to allow the execution of requests scattered over the disk, which could be ignored by SSTF. Both the SSTF and SCAN policies only reduce the seek distance, but are easy to implement by ordering the requests according to their logical block number, assuming the linear mapping from these block numbers to the sector position described earlier. In contrast, incorporating the rotational delay in the scheduling policy requires the exact mapping of the logical block number to the position of the sector on the disk platters to be able to calculate the angle distance. The *Shortest Access Time First* (SATF) [42] policy chooses the request resulting in the shortest overall access time, consisting of both the seek time and rotational delay.

The policies to reduce the access time are often combined with other policies to achieve further goals, such as bounded response times or fairness between the several clients of a storage system. *Aging* policies are used to prioritize requests once they are queued too long, bounding the response times observed by the clients of the storage system [42]. Fair queueing policies, such as the *Completely Fair Queueing* (CFQ) [7], are used to equally distribute the performance of the storage system to its clients. Combining these policies attempts to achieve a compromise between the various requirements posed on a disk-storage system.

2.1.3 Quality-of-Service and Real-Time Requirements

The need to incorporate quality-of-service and real-time guarantees into disk-storage systems is driven by the advent of multimedia applications that process video and audio data [61, 29, 24], commonly referred to as *continuous media*. Compared to traditional systems that typically use textual or binary data [68, 20, 59], the data used by continuous-media applications exhibit a set of different properties.

First and most important, the processing of these continuous-media data requires the timely handling of the data by the storage system. For instance, a video player needs to be able to access a video at

an appropriate frame rate, such as 25 frames per second or one frame each 40 ms, to ensure a glitch-free display of the video. This behavior of continuous-media application results in a periodic access pattern at the storage system. The knowledge of these periodic access patterns, also referred to as *data streams*, allows the storage system to predict the requirements of its clients, and enables the storage system to create a request schedule that fulfills these requirements. The amount of data accessed by the client applications within each period depends on the data type, it can stay constant for uncompressed video or audio data, but it can also vary for compressed video and audio data that are encoded using *variable bit rate* (VBR) algorithms. The period length shown by the access pattern also results from the properties of the data, it can be equal to the frame rate or sampling rate of the data, or it can be larger if the application needs to access several frames or samples at once, such as a *group of picture* of an MPEG video [27]. Second, continuous-media data pose great demands on both the bandwidth required to access the data and the storage size. A DVD-quality MPEG-2 video requires a bandwidth between 4 MBit/s and 8 MBit/s and requires several GByte storage space to store a full movie. The requirements of higher-quality data types, such as HDTV [84], or uncompressed video and audio data are even higher.

Continuous-media applications are not the only cause for the need to ensure real-time requirements in a storage systems. Applications such as real-time transaction systems or interactive media applications also require the timely handling of their storage requests. But in contrast to continuous-media data, theses requests do not necessarily exhibit a periodic access pattern. Instead, individual storage requests must be executed prior to a deadline set by the applications, for instance to ensure adequate response times for interactive applications.

With both continuous-media data streams and interactive applications becoming part of the commodity workload of computer systems, the disk-storage subsystems of traditional workstation operating systems need to incorporate policies to provide the described properties. The storage systems need to provide *quality-of-service* guarantees to their clients. The term quality-of-service commonly refers to a set of properties the storage system needs to provide, including *real-time* guarantees to ensure the timely handling of the data of continuous-media and interactive applications as well as the latency and bandwidth requirements of traditional applications.

2.2 Disk-Storage Systems with Quality-of-Service and Real-Time Guarantees

According to J. Liu [50], computing systems that incorporate timing constraints with the resource management essentially need to provide two properties:

1. A scheduler must execute the applications in an order such that each application achieves its timing constraints. For instance, a CPU scheduler must assign the CPU to the processes of the applications such that each application is able to finish its computation ahead of a deadline. An execution order that achieves this property is called *feasible schedule*.

2. The system must deploy an *admission control* to decide whether new applications can be admitted to the system without interfering with the guarantees of applications already executed by the system. The main element of the admission control is a *schedulability test* that checks whether the scheduler is able to derive a feasible schedule for a given workload.

Applied to disk-storage systems, these properties require the storage system first to incorporate a request scheduling policy that executes disk requests in an order such that the execution of the

requests is finished ahead of the deadlines set by the clients of the storage system (e.g., the display time of the video frame that is read by the disk request). Second, the storage system must use an admission control to decide whether new clients (e.g., data streams) can be accepted without exceeding the abilities of the disk drive. These two requirements can also influence other elements of a storage system, notably the allocation policy of the file system that needs to comply with the resource model used by the admission control.

The remainder of this section will discuss the various approaches to provide these properties with disk-storage systems.

2.2.1 Disk-Request Scheduling

There exists an extensive amount of work that aims to incorporate quality-of-service support into disk-request scheduling, ranging from highly specialized scheduling algorithms used in continuous-media servers to more flexible algorithms that satisfy the diverse requirements of mixed-media systems. In general, the existing approaches can be grouped into the following two categories:

Cycle-based Scheduler

Cycle-based scheduler process disk requests in rounds, executing a sequence of requests in each round for their clients [24, 10, 52, 25]. It is a common approach to comply with the periodic behavior of continuous-media data, often adjusting the round length to the characteristics of the continuous-media data, such as the play time of the data fragments retrieved during each round.

Within each round, several methods exists to schedule the requests of the clients. The simplest approach is to schedule the requests in a *round-robin* manner, executing the requests in a fixed order of the clients. While this approach provides a deterministic behavior with respect to latency, it completely ignores the position of the requests on the disk, which can lead to a high positioning overhead. Executing all requests of a round using the SCAN policy can reduce this overhead, but it increases the maximum latency as all clients have to wait until the end of the period until they can be sure that their requests are executed. The *Grouped Sweeping Scheme* [96, 18] provides a tradeoff between these two approaches, partitioning the round into several groups and assigning the clients to a fixed group. The groups are service in a round-robin order and within a group the requests are executed using the SCAN policy.

The admission control models used with cycle-based schedulers typically limit the number of requests added to each round such that the quality-of-service guarantees are met.

Deadline and Priority-based Scheduler

These schedulers apply the generic real-time scheduling theories [50] to the disk management. With deadline-based schedulers, each disk request gets a deadline assigned that is either directly set by the application or derived from the characteristics of the workload, such as the periodic processing of continuous-media data. The *Earliest Deadline First* (EDF) scheduling policy processes requests in the order of their deadlines, executing the requests with the closest deadline first. To minimize the positioning overhead, the SCAN-EDF policy [64] executes requests with the same deadline according to the SCAN policy. A similar approach exists for the priority-based scheduling [16], executing the requests with the same priority according to the SCAN policy.

In addition to these two generic types of request schedulers, several other approaches for the disk management exist that aim to provide service guarantees, most notably *proportional-share* scheduler [15, 14] and observation-based or feedback-based scheduling policies [90, 95]. Proportional-

share scheduler distribute the disk resource between the clients of the disk system based on predefined weights of the clients. Observation-based scheduler monitor the execution of the disk requests and adapt scheduling parameters if required to ensure service guarantees. Both the proportional-share and observation-based policies provide a more coarse-grained management of the disk resource, and can be combined with low-level disk schedulers. For instance, a proportional-share scheduler can be used to control the amount of requests that are added to the rounds in a cycle-based scheduler.

The concurrent support of different types of quality-of-service and real-time guarantees requires particularly flexible disk-request scheduling policies. Wijayaratne et al. [91, 92] describe an approach to provide different levels of performance guarantees and quality-of-service guarantees for disk I/O with cycle-based request schedulers. The scheduler distinguishes between periodic and aperiodic real-time requests and interactive requests. It uses a two-level scheme that separates the resource scheduling and the bandwidth allocation, using separate admission controllers for each of the request types to limit the number of requests that are scheduled of a type in each round. In case all request are known at the beginning of a round, the bandwidth enforcement of the admission controllers solely ensures that the system achieves all QoS guarantees. However, as both aperiodic requests and interactive requests typically arrive asynchronously at the storage system, the request scheduler deploys the slack time of periodic requests (which are initially ordered using the SCAN policy) to intermingle the execution of the asynchronous requests with the periodic requests.

The *Just-In-Time* (JIT) [56] slack stealing algorithm is an approach to combine the EDF policy to enforce deadlines with the SCAN policy to minimize the positioning overhead as well as with the inclusion of unreserved requests, such as best-effort requests. The slack time of a real-time stream, that is the time the execution of the request of the stream can be postponed without violating its deadline, is calculated whenever a stream is admitted to the system. At runtime, the available slack time of a requests is reduced whenever the execution of the request is postponed in favor of a request closer to the current position of the disk head. If the slack time of a request drops to zero, the request is executed regardless of its location on the disk. The main limitation of this work is that both the slack time and the EDF schedulability test are calculated using only an approximation of the disk utilization caused by the streams. Although this approximation achieves a reasonable utilization of the disk drive, is not sufficient to provide deterministic guarantees.

2.2.2 Mixed-Media Systems

Mixed-media systems combine the requirements of continuous-media systems with the requirements posed by the storage of different data types, such as traditional textual data or digital images [76, 57]. The need for mixed-media systems arose from the advent of advanced applications such as digital libraries and teleteaching that make use of these different data types, as well as from continuous-media and mixed-media applications becoming part of the commodity use of computing systems, which requires commodity systems to comply with the requirements of these applications.

The concurrent support for various data types requires the storage system to combine the different and partially contradicting requirements of those data types. The disk-request scheduler needs to fulfill the timing constraints of continuous-media data as well as requirements such as bounded response times for interactive applications and an overall high throughput. The admission control needs to consider the unpredictability of interactive and best-effort workloads, but is still required to guarantee the timely handling of data streams. Finally, the file system must be able to handle the widely varying space requirements of those data types without causing large overheads, as well as the different access characteristics of the applications using the files.

The following discussion will describe the existing approaches to incorporate the support for mixed-media requirements into disk-storage systems.

Symphony

Symphony [75] is the most comprehensive approach to provide a mixed-media storage system. The design of Symphony addresses the entire range of requirements of a mixed-media system, including the disk-request scheduling, file-system layout and buffer management. It provides an integrated system structure, meaning that the various data types are handled within a single file system, which provides a greater flexibility and better performance compared to a static partitioning of the storage system between the various data types.

The design of Symphony comprises of two layers. First, a data-type specific layer consisting of individual modules that are tailored to the handling of a single data type. Second, a data-type independent layer that provides basic resource-management facilities. The data-type independent layer ensures the secure multiplexing of the storage space, disk bandwidth and buffer space among the different data types, but it still allows the data-type specific modules to implement appropriate policies to manage their data, such as block placement policies and buffer replacement strategies.

To schedule disk requests, the data-type independent layer deploys the *Cello* disk scheduling framework [77]. Cello uses a two-level scheduling policy. At the upper level, Cello distinguishes between several applications classes, where each application class maintains its own request queue that is ordered using a class-specific scheduling policy. At the lower level, a class-independent scheduler multiplexes the available disk resources between the application classes. This class-independent scheduler periodically creates a queue containing the requests scheduled for the current interval, allowing each class-specific scheduler to add requests to this queue. The number of requests a class-specific scheduler is allowed to add to the scheduled queue is governed by a weight assigned to each class. The weights are used by the class-independent scheduler to implement a proportional-share policy, which is either based on the time spent to execute the requests or on the amount of data transfered by the requests.

Cello identifies three generic types of application classes: real-time applications, interactive best-effort applications and throughput-intensive best-effort applications. The requests of real-time application classes are added in SCAN-EDF order to the scheduled queue. The class-specific scheduler for interactive applications aims to provide short response times to these applications. It adds requests to the scheduled queue such that they are executed as early as possible, exploiting the slack time of real-time requests (i.e., the time the execution of a real-time request can be postponed without violating its deadline). The calculation of the slack time of a real-time request is based on finding the latest time the execution of a request must be started in order to meet its deadline. Finally, the class-specific scheduler of throughput-intensive best-effort applications orders its request towards the tail of the scheduled queue using the SCAN or SATF policies to maximize the overall throughput.

Although Symphony provides a comprehensive design of a mixed-media system, it fails to provide a sufficient admission control for real-time streams. In particular, the Cello disk scheduling framework does not include a mechanism to derive the weights of real-time application classes from the requirements of these applications, such as the bandwidth requirements of a data stream. Furthermore, the feasibility of the schedules created by the two-level approach of Cello is affected by the order in that the class-independent scheduler invokes the class-specific scheduler [77], weakening the claim that the two-level approach of Cello cleanly separates the application-independent multiplexing of the disk resources and the application-specific scheduling policies.

Clockwise

The Clockwise storage system [11, 12] combines the support for periodic real-time applications and best-effort applications. It aims to provide short response times to the best-effort applications, but yet to ensure that all deadlines of the real-time applications are met. Clockwise organizes disks using *dynamic partitions*. A dynamic partition is made up of an ordered list of disk blocks that can spread over several disks and that can be dynamically resized. A dynamic partition can either store a single continuous-media file or an entire best-effort file system.

Clockwise employs the ΔL disk scheduler [12]. It schedules the requests of real-time applications based on the EDF policy, and the admission control is based on a non-preemptive version of the EDF schedulability test [43] using worst-case assumptions about the execution times of the disk requests. The admission control additionally calculates the *slack time* ΔL, which is the minimum time between the end of any executed real-time request and its deadline, and that is also calculated based on the non-preemptive EDF schedulability test. Presuming that this slack-time can be applied before the execution of a real-time request, the disk scheduler favors the execution of best-effort requests as long as the execution does not require more time than the available slack time. The best-effort requests are executed in a *first come first served* (FCFS) order.

The ΔL scheduler is able to ensure that real-time applications reach their deadlines and it provides short-response times to best-effort applications, however, both the EDF policy to schedule the real-time requests and the FCFS policy to schedule the best-effort requests do not consider the positions of the blocks accessed by the requests on the disk. The only attempt to improve the disk utilization is to use large block sizes to allocate the dynamic partitions.

USD—User-Safe Disks

The User-Safe Disk (USD) device driver [8, 9] of the Nemesis [47] operating system provides mechanisms to securely multiplex the disk space and disk bandwidth between Nemesis applications. This includes the support for quality-of-service guarantees. USD does not provide a full file-system functionality, instead applications are offered with a stream of requests they can use to implement their own disk management, and USD both enforces quality-of-service guarantees for these streams and ensures that applications do only access the parts of the disk they own.

USD includes two different scheduling policies, the RSCAN algorithm that combines a proportional share allocation of the disk bandwidth with the SCAN scheduling policy and an EDF based scheduling policy. The admission control for both policies is based on worst-case assumptions.

2.2.3 Statistical Real-Time Guarantees

Statistical real-time guarantees are used in *soft* real-time systems to improve the resource utilization for tasks with varying execution times [22, 5, 6, 87]. In contrast to traditional hard real-time systems, systems using statistical real-time guarantees do not ensure that all deadlines are met, exploiting the ability of a number of application classes to tolerate the occasional miss of a deadline. The admission tests used in such systems are commonly based on the calculation of the probability that a task misses its deadline (referred to as *overflow* or *overload* probability) using probabilistic models and admitting tasks as long as this probability stays below an acceptable value. The models used to calculate the overflow probability refrain from the worst-case execution times of the tasks, instead they are based on a more realistic description of the resource usage, such as random variables that describe

the execution times. This approach allows to increase the resource utilization for applications with widely varying execution times.

With the execution times of disk requests varying widely due to the mechanical design of disk drives described earlier in this chapter, statistical guarantees promise to significantly increase the utilization of disk-storage systems. As it will be shown later in this thesis, the worst-case execution time of a disk request can exceed the average case execution time by an order of magnitude. With an admission control that enforces hard real-time guarantees required to be based on worst-case assumptions, this poor ration between average-case and worst-case behavior results in a low utilization of the disk drive for hard real-time guarantees. But especially multimedia applications require the storage system to fully utilize the disk drive to provide the applications with sufficient bandwidths. Moreover, these applications are mostly able to cope with the occasional miss of a deadline or the loss of a data packet, fulfilling the prerequisites for the use of statistical guarantees.

The existing attempts to incorporate statistical guarantees into disk-storage systems focus on continuous-media servers and all use a common approach [89, 97, 58, 44, 17]. Based on the knowledge of the entire workload that is handled by the system (i.e., the number of data streams and the characteristics of the streams), the admission tests calculate the probabilities that the execution of the data streams miss the specified deadlines and that requests of the streams are dropped. The models used to calculate these probabilities are based on random variables describing the amount of data transfered within each round and use either

- random variables describing the aggregated execution time of a number of requests [89],
- a more fine grained execution model of disk requests with random variables describing the individual elements of the model, such as seek time, rotational delay and transfer time [97, 58], or
- fixed overheads to describe the request execution times, limiting the statistical guarantees to the effects of variable-bit-rate streams [17] and caching policies within the server [44].

The admission control accepts new streams as long as the calculated probabilities stay below a threshold, with the threshold either specified globally for all streams [97, 58, 44] or specified independently for each stream [89, 17].

The main shortcoming of this approach is that it requires complete knowledge of the workload in order to calculate the overload probability. This does not allow the request scheduler to easily incorporate requests of clients not accounted by the admission control, such as the requests of sporadic real-time clients or best-effort applications that both cannot be predicted in advance. Furthermore, the request scheduler is not able to drop requests of data streams to limit the quality of the streams in favor of other clients in a mixed-media system. Both incorporating non-stream requests and limiting the quality of the streams to the level accepted by the clients require the explicit knowledge of the execution time required by a stream to achieve its quality, which would allow the request scheduler to reserve this time for the execution of the requests of a stream.

2.2.4 Summary

This section presented an overview of the various approaches to incorporate quality-of-service guarantees into disk-storage systems. Although a large body of previous work exists, none of the approaches is able to provide the entire set of properties required to support quality-of-service guarantees with modern disk-storage systems. In particular, the admission models that make use of statistical

guarantees to improve the utilization of the disk drive lack the support for unaccounted sporadic real-time and best-effort requests. On the other hand, the scheduling mechanisms that respect the diverse requirements of the different data types fail to provide comprehensive admission models, including the support for statistical guarantees.

The most promising approaches to combine diverse quality-of-service requirements are approaches that deploy slack-stealing algorithms [46]. These algorithms utilize the slack time of real-time requests, meaning the time the execution of the request can be postponed without violating its quality-of-service guarantee, to include the execution of unaccounted request such as best-effort requests as well as to apply policies to maximize the disk utilization. To allow the inclusion of statistical guarantees, the request scheduler must be able to derive the slack time of a request such that the statistical guarantees (i.e., the probability that the request misses its deadline) are fulfilled.

The remainder of this chapter will describe the Dresden Real-Time Operating System (DROPS). DROPS includes a reservation-based admission model that incorporates statistical guarantees. This admission model provides the basis to derive an admission model for disk storage that can be used in a slack-time-based request scheduling.

2.3 DROPS—The Dresden Real-Time Operating System

The Dresden Real-Time Operating System addresses the extended needs of advanced real-time applications:

- Applications consist of various parts with varying real-time requirements. For instance, cellphones combine the real-time processing of the communication protocol with multimedia and office-like applications.
- The real-time or quality-of-service requirements of applications differ. Whereas the processing of the cellphone communication protocol or a control application require traditional hard real-time guarantees meaning that all deadlines need to be meet, multimedia applications might tolerate the occasional miss of a deadline but require a high resource utilization.
- Real-time systems often use off-the-shelf hardware not primarily designed to provide a predictable run-time behavior.

DROPS attempts to meet these challenges by providing a flexible system architecture, shown in Figure 2.2. Applications make use of a layered environment consisting of individual software components. Each component provides the applications with a specific service or resource. Components are responsible for managing the resource allocations of applications concurrently using that resource, thus acting as *resource manager*. The resource managers providing complex resources (such as network bandwidth or graphical user interfaces) themselves make use of lower-level resources, forming a hierarchy of resource managers ranging from resource managers handling those complex resources to managers providing basic resources such as CPU time and memory. At the lowest level, DROPS uses the FIASCO microkernel [36], an implementation of the L4 microkernel API [48] tailored to support real-time systems. FIASCO provides the mechanisms used by the resource managers to ensure the resource isolation between their clients, including the reservation-based CPU scheduling mechanism described later in this section.

One of the main properties of DROPS is the coexistence of real-time and non–real-time subsystems. The primary non–real-time subsystem consists of L^4LINUX [34], a paravirtualized version of the Linux kernel. Both subsystems can interact in various ways, L^4LINUX can make use of services provided by

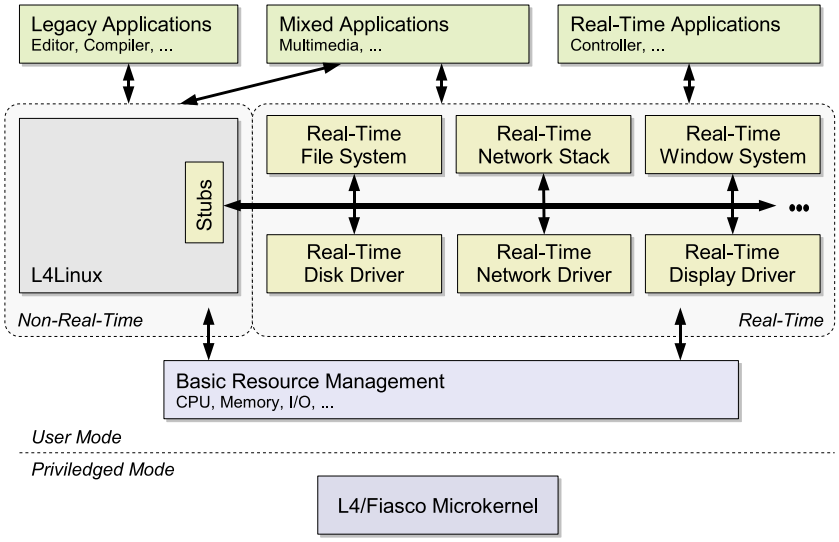


Figure 2.2: DROPS Overview.

the real-time subsystem, such as a network driver (through a stub driver) and real-time applications can move parts that have no real-time requirements (e.g., user interfaces) to L^{4}LINUX, utilizing the fully-featured Linux environment. This approach highlights a major design principle of DROPS, splitting applications into a part with real-time requirements and another part without such requirements, and providing proper environments for both parts.

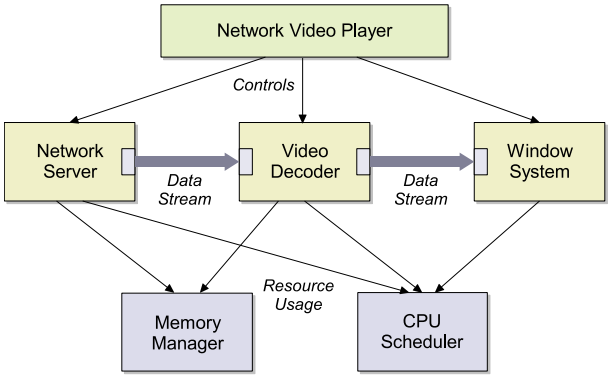


Figure 2.3: DROPS Application Model.

Figure 2.3 illustrates the general application model used in DROPS. An application consists of several components processing data streams. Components allocate required resources at resource managers and are resource managers themselves providing a complex resource to the application [39].

2.3.1 Quality-Assuring Scheduling—QAS

An essential element of the DROPS architecture is the admission model deployed by the resource managers to schedule active resources. The *Quality-Assuring Scheduling* (QAS) [31]

- provides a deterministic system behavior in case of overload situations, and
- improves the resource utilization by using a statistical admission model instead of worst-case assumptions.

QAS achieves these goals by splitting periodic tasks into a mandatory and one or more optional parts. Mandatory parts have to be executed under any circumstances. Optional parts may be dropped in case of resource shortage. However, QAS ensures that a minimum number of optional parts are successfully executed over a longer period of time. To assure this minimum quality, the scheduler assigns the required resource amount, referred to as *reservation time*, to the optional parts. The reservation time is calculated based on a probabilistic model using random variables describing the resource demand of the task parts.

The resource managers enforce the reservation times. An optional part is not allowed to consume more resources within a period than it got assigned with the reservation time. This prevents optional parts from consuming more resources than required to achieve their qualities, allowing the execution of parts with a lower priority even in case of resource shortage.

The remainder of this section provides a formal description of the QAS admission model.

Task Model

Each task T_i of a task set $T = \{T_1, \ldots, T_n\}$, $n \in \mathbb{N}^+$ is a sequence of jobs J_{ij}, $j \in \mathbb{N}^+$, to be processed periodically (Fig. 2.4). Each job J_{ij} consists of a mandatory part M_{ij} and c_i optional

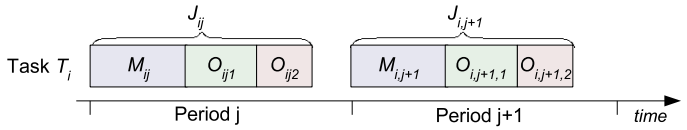


Figure 2.4: QAS Task Model. The jobs of a task are divided into a mandatory part and one or more optional parts.

parts O_{ijk}, $k = 1, \ldots, c_i$; the number c_i of optional parts in a period is fixed for each task. The resource demand of both the mandatory parts and of the optional parts are described by independent, nonnegative random variables X_i and Y_{ik}, $k = 1, \ldots, c_i$, respectively. Finally, an application may specify probabilities q_{ik} defining the requested qualities of its optional parts.

The following definition is a general description of a task in the QAS model:

Definition. A task T_i is a tuple

$$T_i = (X_i, w_i, c_i, Y_{i1}, \ldots, Y_{ic_i}, q_{i1}, \ldots, q_{ic_i}, d_i), \qquad i \in \mathbb{N} \tag{2.1}$$

where

X_i nonnegative random variable, resource demand of the mandatory parts M_{ij};

w_i nonnegative real number, worst-case resource demand of the mandatory parts M_{ij}, that is $\mathbf{P}(X_i \leq w_i) = 1$;

c_i nonnegative integer, number of optional parts;

Y_{ik} nonnegative random variable, resource demand of the k^{th} optional part;

q_{ik} real number $0 \leq q_{ik} < 1$, requested quality of the k^{th} optional part;

d_i positive real number, length of the period and relative deadline for the execution of both the mandatory and optional parts.

Note that setting $X_i \equiv 0$ describes a task consisting of optional parts only.

The following descriptions will abstract from viewing at an individual period, identifying the parts with their random variables. Each mandatory part M_{ij} is considered a realization of the random variable X_i and an optional part O_{ikj} is considered a realization of Y_{ik}.

Admission Control

The aim of the admission control is to derive the scheduling parameters of a task set, namely the priorities of the mandatory and optional parts and the reservation times of optional parts. To successfully admit a task set, the admission control must be able to ensure a feasible schedule. Such a schedule must fulfill the following two prerequisites:

1. All mandatory parts X_i must meet their deadlines. The deadline of a part is the end of the period. In case of a task set with uniform periods (i.e., $d_1 = \cdots = d_n =: d$) this holds if and only if

$$\sum_{i=1}^{n} \frac{w_i}{d} \leq 1, \qquad n : \text{number of tasks in } T \tag{2.2}$$

2. The reservation times r_{ik} can be found such that all optional parts Y_{ik} achieve the requested quality q_{ik}.

   Let $p_{ik}(r)$ denote the probability that the optional part Y_{ik} is successfully executed (i.e., meets its deadline). Then, the reservation time r_{ik} is the smallest time r where $p_{ik}(r)$ is at least the requested quality q_{ik}:

$$r_{ik} = \min(r \in \mathbb{R} | p_{ik}(r) \geq q_{ik}) \qquad \forall i = 1, \ldots, n; k = 1, \ldots, c_i \tag{2.3}$$
$$n : \text{number of tasks in } T$$

These two conditions define the general admission criterion for a task set T.

It is important to note that the calculation of $p_{ik}(r)$ uses the random variables to describe the resource demands for *both* the mandatory parts and the optional parts. Thus, optional parts might be admitted to a task set even if the mandatory parts of the task set already fully utilize the resources based on their worst-case resource demands.

To assign the scheduling priorities to the mandatory and optional parts, QAS introduces the *Quality-Monotonic Scheduling* (QMS). By analogy to the Rate-Monotonic Scheduling (RMS) [49], QMS assigns the priorities based on the requested quality of the parts. For task sets with uniform period lengths,

the priorities of mandatory parts are higher than the priorities of optional parts and the higher the quality of an optional part, the higher its priority:

$$\left.\begin{array}{lcll} pr(Y_{ik}) & \succ & pr(Y_{jk}) & \text{if} \quad q_{ik} > q_{jk} \\ pr(X_i) & \succ & pr(Y_{jk}) & \end{array}\right\} \quad \forall i,j = 1,\ldots,n; k = 1,\ldots,c_i \tag{2.4}$$

where $pr(X_i)$ and $pr(Y_{ik})$ denote the priorities of a mandatory part and an optional part respectively, and $\succ$ means higher for priorities. For task sets consisting of several period lengths, the task set is decomposed into m disjoint subsets S_i, $i = 1,\ldots,m$, where a subset consists of all tasks with the same period length d_i. The subsets are ordered according to the period length, starting with the shortest period. Within the tasks of a subset, the priorities are assigned as described for uniform periods, but additionally ensuring that all priorities assigned to task parts of a task set with a shorter period are higher than the priorities assigned within a task set with a longer period:

$$\underline{pr}(S_i) \succ \overline{pr}(S_j) \quad \text{if} \quad d_i < d_j \qquad \forall i,j = 1,\ldots,m \tag{2.5}$$

where $\underline{pr}(S_i)$ denotes the lowest priority assigned within a task set and $\overline{pr}(S_i)$ the highest priority.

This priority assignment is assumed to be optimal with respect to feasibility, meaning that if a feasible schedule does not exist under QMS then such a schedule does not exist under any other fixed priority assignment. However, an exact proof of this assumption could not be found until now, mainly due to the structure of the admission criterion.

2.3.2 CPU Scheduling in DROPS

CPU scheduling in DROPS consists of two parts, a scheduling mechanism provided by the FIASCO microkernel and a user-level scheduler that particularly executes the admission control.

FIASCO Kernel Scheduling

L4 employes a fixed-priority round-robin scheduling. The kernel schedules threads with the highest priority until they yield the CPU by a blocking IPC operation. Threads with the same priority are scheduled in a round-robin manner with an allocated timeslices to determine the amount of time each thread is allowed to run without being preempted.

FIASCO extends the basic L4 scheduling model with the support for periodic real-time threads [82]. Real-time threads are characterized by three additional properties:

1. a period length (deadline);
2. one ore more *reservation scheduling contexts*, such a context consist most notably of a priority and a time quantum; and
3. a best-effort scheduling context containing the non–real-time priority and time quantum of the thread.

For threads executing in real-time mode, the kernel periodically assigns the reservation contexts to these threads, starting with the first context. Thus, the kernel sets the scheduling priority of the thread to the priority stored in the scheduling context, executing the thread on the priority-level of the scheduling context[1].

[1]The final scheduling policy is still based on the priorities of the threads. Thus, a non–real-time thread can preempt a real-time thread if the priority of the non–real-time thread is higher than the priority specified in the reservation scheduling context of the real-time thread.

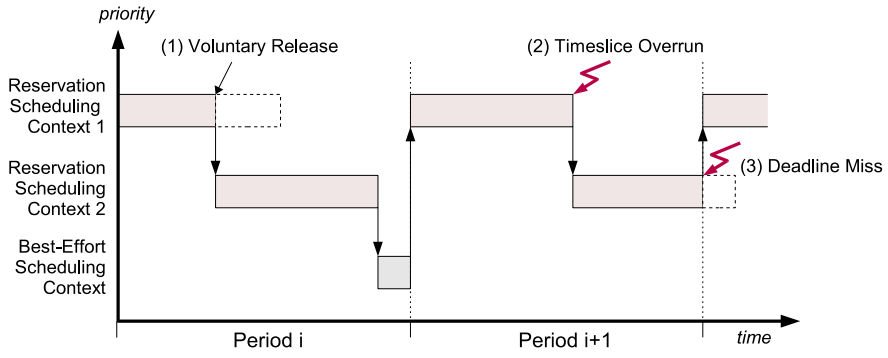


Figure 2.5: FIASCO Real-Time Scheduling

The switch to the next scheduling context occurs in various situations, illustrated in Figure 2.5:

1. the thread voluntarily releases the scheduling context,
2. the thread exhausts the time quantum specified in the scheduling context, or
3. the thread reaches the end of the period.

In the latter two cases the kernel generates a preemption IPC message to the scheduler of the thread, allowing the application to respond to the reservation overrun or deadline miss. Once a thread reached its last reservation scheduling context, the kernel switches to the best-effort scheduling context of the thread, allowing the thread to continue its execution in case no other thread (real-time or non–real-time) is active with a higher priority.

FIASCO supports various modes to release real-time threads. In the strictly-periodic mode, the kernel immediately releases the thread (i.e., sets the effective priority of the thread to the priority specified in the first scheduling context and replenishes timeslice with the time quantum) at the beginning of a new period. In the periodic mode, the kernel additionally waits for an external event, which is sent to the thread by an IPC. These modes allow the implementation of a variety of scheduling schemes, including strictly-periodic threads, threads with a minimum inter-release time and deferred servers [50].

User-level Scheduler / Admission Server

The admission server is responsible for admitting new applications to the system and assigning the reservation scheduling contexts to the associated threads. This particularly includes the calculation of the reservation times of optional parts according to Condition 2.3. To admit mixed task sets consisting of both tasks described by QAS parameters and traditional periodic tasks described by $T_i = (p_i, C_i, D_i, t_i)$[2], the admission server combines the QAS model with an online time-demand analysis [50].

[2] $T_i = (p_i, C_i, D_i, t_i)$ with p_i denoting the static priority, C_i worst-case execution time, D_i the deadline, and T_i the period length of task T_i.

2.4 Summary

This chapter presented the foundations of this thesis and discussed related approaches to provide quality-of-service guarantees in disk-storage systems.

The remainder of the thesis will describe and evaluate the design of a disk storage system that supports diverse quality-of-service requirements. The admission control is based on the concepts of the Quality-Assuring Scheduling, calculating reservation times for data streams such that the streams achieve statistical guarantees. A slack-stealing request scheduler enforces the reservation times, utilizing the time not assigned to data streams to include sporadic real-time and best-effort requests as well as to maximize the utilization of the disk drive.

Chapter 3

Disk Storage with Quality-of-Service Guarantees

The disk-storage subsystem of a multi-service system such as DROPS has to support applications with different service requirements:

- Continuous-media applications, such as video and audio streaming, require bandwidth guarantees to ensure a glitch-free display of the data.
- Individual deadlines for disk requests are required to support applications with guaranteed-latency response times, such as real-time database or interactive multimedia applications.
- Best-effort applications raise demands such as high disk throughput, short response times and fairness between applications.

These demands are not isolated but occur concurrently in one system.

To support these diverse quality-of-service requirements, a disk-storage system must apply a flexible scheduling mechanism that multiplexes the disk resource between the clients of the storage system such that each client achieves its quality-of-service demands [61]. The scheduling mechanism must include an admission control that governs the number of clients accepted by the system such that the workload does not exceed the bandwidth capacity of the disk. Once clients are admitted to the storage system, a disk-request scheduler must ensure that the disk requests of the clients are executed in an order such that all clients achieve their quality-of-service demands, as well as that the utilization of the disk drive is optimized.

The following chapter describes the disk-scheduling model that incorporates the support for diverse quality-of-service requirements. Section 3.1 outlines the basic model used to describe the disk resource. Section 3.2 describes the admission control that follows the ideas of the Quality-Assuring Scheduling, and Section 3.3 describes the disk request scheduling mechanism used to enforce the service guarantees.

3.1 Resource Model

For an admission control to be able to decide whether a new client can be admitted to the system, it requires the knowledge of the resource demands of the clients. Clients typically specify their resource

demands in terms of high-level properties such as the bandwidth of a data stream. These high-level properties must then be mapped to a resource model that both matches the execution model of the storage system and can be used in a formal schedulability test.

3.1.1 Data Streams

Continuous-media applications require the timely delivery of their data by the storage system to avoid glitches within the processing of the data streams. The elements of a stream, such as video frames or audio samples, must be available to the application prior to their play time. Similarly, for continuously recording applications, such as video or audio recording or the data gathering of scientific applications, the storage system must ensure the timely writing of data to disk.

To ensure the continuous execution over time, the disk storage system has to provide an sufficient bandwidth to the application. The required bandwidth is a property of the data stream. For instance, for video streams it is defined by the frame size and the frame rate of the video. This bandwidth requirement defines the resource demand at the client level of the storage system, and it must be translated to a resource specification that can be used by the storage system to schedule this resource demand.

At the disk level, the storage system executes individual disk requests to read data from the disk or to write data to the disk. The bandwidth that is achieved by the storage system depends on the number of requests executed within a time span and the size of the data transfered by these requests:

$$bw = \frac{b_1 + b_2 + \cdots + b_a}{d}, \tag{3.1}$$

where b_i denotes the amount of data transfered by a request and d specifies the time required to execute all a requests. To derive a reasonable resource model that can be used in an admission control, requests originating from data streams are executed periodically using a fixed request size b. This allows to describe the resource requirement of a data stream in terms of a fixed number of disk requests that have to be executed periodically. Thus, the resource demand is defined by a tuple

$$R = (a, b, d), \tag{3.2}$$

where a denotes the number of requests executed within a period of length d, and b denotes the request size. The bandwidth achieved by a data stream described by this tuple is defined by:

$$bw = \frac{a \cdot b}{d}. \tag{3.3}$$

To provide a client with a requested bandwidth bw_{Stream}, the storage system needs to derive an appropriate combination of request size, number of requests, and period length to achieve this bandwidth. With the required number of requests resulting from the request size and period length,

$$a = \left\lceil \frac{bw_{Stream} \cdot d}{b} \right\rceil, \tag{3.4}$$

suitable values for the request size and period length need to be set by the storage system. To derive these values, the storage system needs to consider the influence of both of these values on the overall system behavior as well as limitations caused by the structure of the storage system:

- The request size must match the allocation of the stream data on disk. To ensure that each request of the stream can be executed with the request size b, the request size must not exceed

the block size used to allocate the stream data on the disk. More precisely, the request size $b_{Request}$ must be a divisor of the allocation block size $b_{Allocation}$:

$$b_{Request} | b_{Allocation} \tag{3.5}$$

This property not only creates a limitation on the possible request sizes, but it also poses a requirement for the allocation policy used with the file system. The file system needs to ensure that the streams are stored using the block size $b_{Allocation}$ or a multiple thereof.

- The ratio of the request size and period length sets the granularity of the bandwidth allocation. The smallest unit of bandwidth that can be allocated is defined by the execution of a single disk request. The portion of bandwidth contributed by a single disk request is

$$bw_{Request} = \frac{b}{d}. \tag{3.6}$$

Thus, a large request size or a short period result in a coarse-grained bandwidth allocation, whereas a longer period or a smaller request size allow for more fine-grained bandwidth assignments.

- The request size significantly influences the utilization of the disk drive; by increasing the request size a better disk utilization can be achieved.

- The period length influences both the buffer requirement and the latency of a data stream. With the release time of the requests defined by the begin of a period and the deadline defined by the end of a period, the period length defines the maximum time required to respond to events such as the start of a stream or the repositioning with a data stream. Also, sufficient buffer memory must be available to store the data of at least one period.

These constraints necessitate the flexibility to choose appropriate values for the request size and the period length depending on the demands of the individual data stream. To reduce buffer size of a data stream with a high bandwidth requirement, a short period length should be used, whereas a longer period length should be used for data streams with a smaller bandwidth requirement to allow the use of large block sizes.

Variable Bit-Rate Streams

The preceding discussion holds for data streams with constant bandwidth requirements. However, many compression algorithms for video and audio data use variable bit rate (VBR) encoding. Thus, the resulting bandwidth requirements depend on the dynamics of a video scene or audio fragment. Considering this variation in the admission control, for instance by multiplexing the data streams in a way such that the overall bandwidth requirement of the set of data streams is smoothed, leads to a complex admission model. Instead, I chose to use a buffering scheme to compensate the variance within each data stream, enabling a constant bandwidth allocation at the disk storage system. Using buffering to smooth the bandwidth variations of VBR streams is a common technique in the context of the delivery of video streams over networks [54, 70]

Obviously, the storage system should deliver the data stream at least with the average bandwidth required by the client. However, this can lead to high buffer requirements for bursty data streams. To reduce the buffer requirement, the data stream can be delivered with a higher bandwidth, allowing a tradeoff between memory consumption and bandwidth requirements. To avoid buffer overruns, which may occur with a higher bandwidth allocation, the disk storage system should be able to skip

requests if it detects that the buffer contains sufficient data to ensure the contiguous delivery of the data. The required amount of buffer can be calculated based on the delivery bandwidth of the storage system and either a function describing the actual consumption over time [33] or a stream model that characterizes the burstieness of the data stream [30, 32].

3.1.2 Non–Data-Stream Traffic

Although the bandwidth requirement described earlier is the most common form of quality-of-service requirements requested from a disk storage system, not all application requirements can be directly mapped to the bandwidth model. Applications such as real-time transaction systems require a guaranteed response time of a particular disk request. With respect to the general real-time scheduling theory, these applications form *sporadic* or *aperiodic* tasks [50]. There exist two approaches to include such tasks in the admission control:

- The task can be mapped to a strict periodic execution model if the requests yield a reasonable minimal interrelease time.
- A portion of the bandwidth can be reserved in the admission through a special data stream and the jobs of such tasks can be mapped to that data stream once they are released.

Even in the case that sporadic or aperiodic tasks can not be included in the admission control, an acceptance test can be used at runtime to decide whether a disk request can be admitted to the system based on the current utilization of the disk. The acceptance test ensures that if a request is admitted to the system it meets its deadline. However, no guarantees can be given that a request is admitted to the system at all.

3.1.3 Resource Specification

To summarize, clients specify their resource demand in terms of a bandwidth bw and optionally a block size b or a period length d:

$$D = (bw, b, d) \tag{3.7}$$

If an application omits the specification of the block size or the period length, the disk system is free to choose an appropriate value. From this specification, the disk system derives the resource specification R that is used by the admission control:

$$R = (a, b, d) \tag{3.8}$$

with a denoting the number of disk requests of size b that are executed in a period of the length d. a is defined by:

$$a = \left\lceil \frac{bw \cdot d}{b} \right\rceil \tag{3.9}$$

The following section derives an admission model that bases on this resource specification.

3.2 Quality-Assuring Disk Scheduling

As already motivated, an admission model used in a disk-storage system should incorporate statistical quality-of-service guarantees. This requirement results from the observation that on the one hand hard real-time guarantees based on worst-case assumptions lead to a poor utilization of the disk; and that on the other hand the applications that make use of the storage system often are capable to tolerate the occasional miss of a deadline or the loss of a data packet.

A popular example of how the latter property is utilized in a different context is the streaming of video and audio data over a network. The applications are able to resynchronize to the data stream if a network packet is received to late or is lost completely [23, 13]. But both N. Feamster et al. [23] and J.M. Boyce et al. [13] also show that the effect of a packet loss on the quality of a video display depends on the part of the data stream where the loss happened. Video compression algorithms such as MPEG-1/2 and MPEG-4 use correlations between frames to achieve high compression levels. This leads to a propagation of an error to subsequent frames if a data loss happened on a reference frame (e.g., an *I-Frame* in an MPEG stream), whereas packet loss on a frame with no references to it only affects the single frame. This property of video compression assigns several levels of importance with respect to the display quality to the different parts of a data stream. The storage system must respect these varying quality-of-service requirements within a stream. Applications should be able to request that important parts of the data stream, such as the reference frames of a video or indexing information, are delivered with a higher quality than other parts of the stream.

The partitioning of a data stream into parts with different quality requirements follows the high-level structure of the data stream, for example the frame structure of an MPEG video stream. With the data stream model defined in Section 3.1, which requires fixed-sized disk requests, a single request can cover parts of the data stream with different quality requirements. To respect these multiple quality requirements, an admission-control model would need knowledge about the high-level structure of the data streams, which would limit the applicability of such a model only to known types of data streams. To make the admission model independent of the high-level data structure, data streams that contain parts with varying quality requirements should be instead split into substreams with fixed quality requirements, allowing the admission model to consider the substreams separately. For an MPEG stream this approach means that the important parts, such as indexing information and reference frames, are moved to a stream with a high quality requirement, and less important parts are moved to a stream with a lower quality requirement (illustrated in Fig. 3.1).

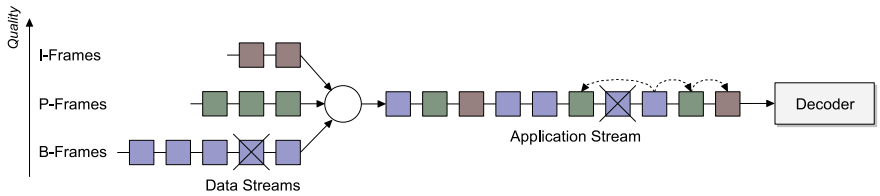


Figure 3.1: Splitting an MPEG-2 video. The different frame types of the MPEG-2 video are stored and delivered in separate data streams. To display the video, the streams are joined again to provide the decoder application with the expected frame order. Requiring high qualities for the streams that contain reference frames makes sure that these frames are almost always available to the decoder, avoiding that a loss of a single frame affects a whole group of picture.

To derive an admission model that respects the varying quality requirements of the substreams, one can apply the Quality-Assuring Scheduling model described in Section 2.3.1 in the following way. A task T_i describes a data stream, and instead of identifying mandatory and optional parts within

a task, the quality requirement is a property of the whole task. A mandatory task (i.e., a task with a quality requirement of 1) identifies a data stream for that the storage system needs to make sure that all requests are executed successfully. Optional tasks describe data streams for that the storage system must ensure that at least a requested percentage of requests, specified by the stream quality, is successfully executed.

The remainder of this section provides a detailed discussion of the Quality-Assuring Disk Scheduling, starting with a description of the modified task model, followed by the formulation of the precise schedulability test. The Quality-Assuring Disk Scheduling assumes a priority-driven, reservation-based disk-request scheduling mechanism, that will be described in Section 3.3.

3.2.1 Task Model

The modifications of the general QAS model result in changes to the task model set by Definition 2.1:

- A task T_i describing a data stream consists of just one part. Such a part consists of several disk requests, both the number a_i of disk requests of a part and the request size b_i of these disk requests are fixed, as discussed in Section 3.1.
- A random variable X_i specifies the time required to execute a *single* disk request. Likewise, w_i specifies the worst-case execution time of a single disk request. Both X_i and w_i depend on the request size b_i used by the data stream.
- An application specifies a probability q_i for each task, $q_i = 1$ defines a mandatory task meaning that all disk requests of that data stream must be executed. $0 \leq q_i < 1$ defines an optional data stream.

Altogether, the following task definition describes a data stream:

Definition. A task T_i is a tuple

$$T_i = (a_i, X_i, w_i, q_i, d_i) \qquad i = 1, \ldots, n,\ n \in \mathbb{N} \tag{3.10}$$

where

a_i positive integer, number of disk requests executed in each period;

X_i nonnegative random variable, execution time of a single disk request;

w_i positive real number, worst-case execution time of a single disk request;

q_i real number $0 \leq q_i \leq 1$, requested quality of the data stream;

d_i positive real number, period length.

This task model to describe a data stream is similar to the task model used by the *Statistical Rate Monotonic Scheduling (*SRMS*)* [6] algorithm. However, there are three main differences:

1. Disk requests cannot be preempted once they are started,
2. a task consists of several subjobs, and
3. the execution time of a disk request is not known at the release time of the request.

All of these properties need to be considered by the admission control, which is described next in this section. Throughout this discussion, the terms task and data stream are used interchangeably.

3.2.2 Admission Control

In order to successfully admit a task set, the admission control needs to derive the scheduling parameters that produce a feasible schedule for that task set. These scheduling parameters comprise the task priorities that define the execution order of the tasks and the reservation times.

Based on the Quality-Monotonic Scheduling described in Section 2.3.1, tasks are ordered according to their quality and period length, the shorter the period of a task, the higher is the task's priority and tasks with the same period length are ordered according to their quality:

$$\left.\begin{array}{lcll} pr(T_i) & \succ & pr(T_j) & \text{if} \quad d_i < d_j; \quad \text{and} \\ pr(T_i) & \succ & pr(T_j) & \text{if} \quad (d_i = d_j) \wedge (q_i > q_j) \end{array}\right\} \quad \forall i, j = 1, \ldots, n \tag{3.11}$$

Figure 3.2 depicts an example of the ordering scheme.

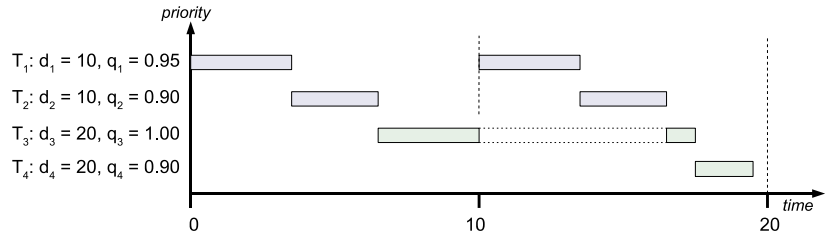


Figure 3.2: QAS Task Ordering

As described in Section 2.3.1, the admission control needs to ensure that both all mandatory parts meet their deadline and that reservation times are found such that all optional parts achieve the requested quality. Applied to disk data streams, this requirement means that all requests of data streams with $q_i = 1$ must be executed within a period. With the quality q_i of an optional stream defined as the percentage of successfully executed disk requests of that stream, the admission criterion for an optional data stream is defined by

$$\exists r_i : \mathbf{E}A_i(r_i) \geq a_i \cdot q_i \qquad \forall i = 1, .., n \tag{3.12}$$

where $A_i(r)$ is a random variable describing the number of executed disk requests for a data stream depending on the reservation time r and $\mathbf{E}A_i(r_i)$ denoting the expected value of that random variable. To successfully admit a task set, the admission control needs to solve the system of equations defined by Equation 3.12.

The remainder of this section describes the approach to solve this system of equations, starting with the discussion of the solution for task sets consisting of uniform periods, and generalizing the approach to task sets consisting of harmonic periods afterward.

Admission Control for Uniform Periods

For uniform periods (i.e., $d_1 = \cdots = d_n =: d$) the elements of a task set are solely ordered according to their quality. Thus, the first m tasks of a task set describe mandatory data streams, with m being the number of mandatory data streams. To admit these mandatory data streams, the condition

$$\sum_{i=1}^{m} a_i \cdot w_i \leq d \tag{3.13}$$

must hold. Additionally, the admission control calculates the random variable X, which denotes the aggregated execution time of the disk requests of all mandatory streams. It is defined by

$$X = \sum_{i=1}^{m} \sum_{j=1}^{a_i} X_i. \tag{3.14}$$

The summation of X_i requires the convolution of the distributions that describe X_i.

For the optional data streams $T_{m+1} \dots T_n$, the admission control must solve the system of equations defined by Formula 3.12. The random variable $A_i(r)$ consists of the integer elements $0, \dots, a_i$, with $A_i(r) = 0$ meaning that no request of the i^{th} data stream is executed in a period, $A_i(r) = 1$ that one request is executed, and so on.

Figure 3.3 outlines the principle approach to calculate the probabilities $\mathbf{P}(A_i(r) = k)$ (i.e., the distribution law of $A_i(r)$) for a given reservation time r, showing the situation for the first optional data stream executed after all mandatory streams have been executed (X denoting the aggregated execution time of the mandatory streams). The figures illustrate the two cases where the execution of disk requests (the requests are denoted by Y_j, meaning a realization of the random variable X_i that describes the execution time) of the stream stops:

1. The stream exhausted its reservation time, shown by Figure 3.3(c). After the execution of the request Y_3 no further request of the stream is executed.

2. The stream reached the end of the period, shown by Figure 3.3(f). The request Y_3 is not executed, although the stream did not yet exceeded its reservation time.

These two conditions must be combined to calculate the probability that the k^{th} requests of the $(m+1)^{th}$ stream is executed:

$$\begin{aligned} \mathbf{P}(A_{m+1}(r) \geq k) &= \mathbf{P}(\overbrace{X + \sum_{j=1}^{k-1} Y_j < d}^{(1)} \ \wedge \ \overbrace{\sum_{j=1}^{k-1} Y_j < r}^{(2)}), \qquad k = 2, \dots, a_{m+1} \\ \mathbf{P}(A_{m+1}(r) \geq 1) &= \mathbf{P}(X < d) \end{aligned} \tag{3.15}$$

The first part (1) of the term specifies the condition that the execution did not reach the end of the period. The second part (2) specifies the condition that the stream did not exceed its reservation time. As the execution times of the disk requests Y_j of a data stream are identically distributed described by the random variable X_{m+1}, the sum $\sum_{j=1}^{k-1} Y_j$ that describes the aggregated execution time of the first $k-1$ requests of the data stream can be replaced with the $(k-1)$-times summation of X_{m+1}:

$$(k-1)X_{m+1} = \sum_{j=1}^{k-1} X_{m+1}. \tag{3.16}$$

Using this definition, the distribution law of the random variable A_{m+1} can be calculated as follows:

$$\begin{aligned} \mathbf{P}(A_{m+1}(r) = a_{m+1}) &= \mathbf{P}(A_{m+1}(r) \geq a_{m+1}) \\ \mathbf{P}(A_{m+1}(r) = k) &= \mathbf{P}(A_{m+1}(r) \geq k) - \mathbf{P}(A_{m+1}(r) \geq k+1), \qquad k = 1, \dots, a_{m+1} - 1 \\ \mathbf{P}(A_{m+1}(r) = 0) &= \mathbf{P}(X \geq d) \end{aligned} \tag{3.17}$$

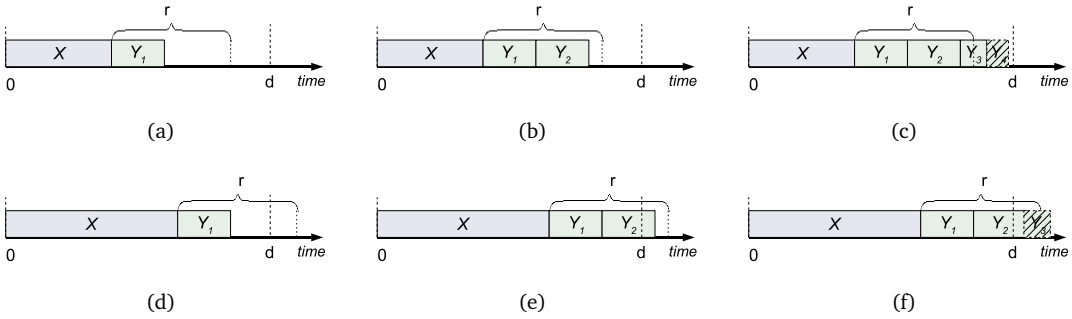


Figure 3.3: Calculation of the random variable $A(r)$. Figures (a)–(c) show the cases where the execution of requests is stopped because the stream exhausted its reservation time; Figures (d)–(f) where the execution is stopped because the stream reached the end of the period

Based on $A_{m+1}(r)$, the admission control determines the reservation time r_{m+1} by finding the smallest value of r where the data stream achieves the requested quality as defined by Equation 3.12:

$$r_{m+1} = \min(r \in \mathbb{R} | \mathbf{E}A_{m+1}(r) \geq a_{m+1} \cdot q_{m+1}) \tag{3.18}$$
$$\mathbf{E}A_{m+1}(r) = \sum_{k=0}^{a_{m+1}} k \cdot \mathbf{P}(A_{m+1}(r) = k)$$

The approach presented so far does not consider an important corner case, illustrated by Figure 3.3(e). Although the conditions defined earlier ensure that no request is executed once the execution reaches the end of the period, the actual execution of the last request of a period can overlap with the new period, causing a delay of the execution of the mandatory streams of the next period. This case is an example for *blocking due to nonpreemptivity* [50] and can be resolved by considering the maximum blocking time in the admission control of both mandatory and optional data streams. The maximum blocking time is defined by the worst-case execution time of a disk request. Thus, using $d' = d - w_{max}$ (with w_{max} being the maximum worst-case execution time w_i of a task in the task set)[1] instead of d ensures an accurate admission control, preventing the execution of requests overlapping with the following period.

To subsume, the reservation time r_{m+1} of the first optional part is calculated by deriving the smallest time r where the data stream achieves the requested quality, denoted by the expected value of the random variable $A_{m+1}(r)$. The random variable $A_{m+1}(r)$ is calculated based on the execution times of the requests of the stream (Y_j) and aggregated execution time of the mandatory streams (X). To generalize this approach to the calculation of the reservation times of the remaining optional streams, one needs to be able to derive the aggregated execution time of all streams preceding a given optional stream T_i, which then allows the calculation of $A_i(r)$ just as described for the first optional stream.

As already described, for a mandatory stream T_i the aggregated execution time Z_i after the execution of this stream results from

$$Z_i = Z_{i-1} + \sum_{j=1}^{a_i} X_i \qquad \forall i \leq m \tag{3.19}$$

which denotes the summation of the execution times of the individual requests of the stream T_i with the aggregated execution time of the streams $T_1, \ldots, T_{i-1}$. Z_0 is defined by a random variable with $\mathbf{P}(Z_0 = 0) = 1$. For an optional stream T_i, the calculation of Z_i is also based on a summation of the

[1] The complete approach would be to calculate d' for each data stream separately, $d'_i = d - w_{max_i}$ with $w_{max_i} = max(w_j), j = i+1, \ldots, n$. However, for simplicity reasons I assume the worst-case execution time of all disk request to be fixed.

execution times of the requests of that stream with Z_{i-1}, but the summation must consider that not necessarily all requests of an optional stream are executed:

$$Z_i = Z_{i-1} + \sum_{j=1}^{a_i} \begin{cases} X_i & \text{if } \sum_{k=1}^{j} X_i < r_i \ \wedge \ Z_{i-1} + \sum_{k=1}^{j} X_i < d' \\ 0 & \text{otherwise} \end{cases} \qquad \forall i = m+1, \ldots, n \tag{3.20}$$

The exact approach to solve this equation will be discussed later in this section.

Using this definition of Z_i, the general approach to calculate the reservation times r_i for the optional streams is then defined by:

$$r_i = \min(r \in \mathbb{R} | \mathbf{E}A_i(r) \geq a_i \cdot q_i) \qquad \forall i = m+1, \ldots, n \tag{3.21}$$
$$\mathbf{E}A_i(r) = \sum_{k=0}^{a_i} k \cdot \mathbf{P}(A_i(r) = k)$$

with

$$\begin{aligned} \mathbf{P}(A_i(r) = a_i) &= \mathbf{P}(A_i(r) \geq a_i) \\ \mathbf{P}(A_i(r) = k) &= \mathbf{P}(A_i(r) \geq k) - \mathbf{P}(A_i(r) \geq k+1), \qquad k = 1, \ldots, a_i - 1 \\ \mathbf{P}(A_i(r) = 0) &= \mathbf{P}(Z_{i-1} \geq d') \end{aligned} \tag{3.22}$$

where

$$\begin{aligned} \mathbf{P}(A_i(r) \geq k) &= \mathbf{P}(Z_{i-1} + (k-1)X_i < d' \ \wedge \ (k-1)X_i < r), \qquad k = 2, \ldots, a_i \\ \mathbf{P}(A_i(r) \geq 1) &= \mathbf{P}(Z_{i-1} < d') \end{aligned} \tag{3.23}$$

The admission control admits a task set only if it can solve the Equation 3.21 for all optional data streams, meaning that the admission control finds reservation times such that all data streams achieve their requested quality.

Admission Control for Harmonic Periods

With multiple periods, the streams with shorter periods have higher priorities, streams with the same period are ordered according to their quality. This priority scheme results in an execution order of the data streams as shown in Figure 3.4. The requests of the data streams with the shortest period are executed with the highest priority, requests of data streams with a longer period are executed if the data streams with the shortest period did not exhaust their period.

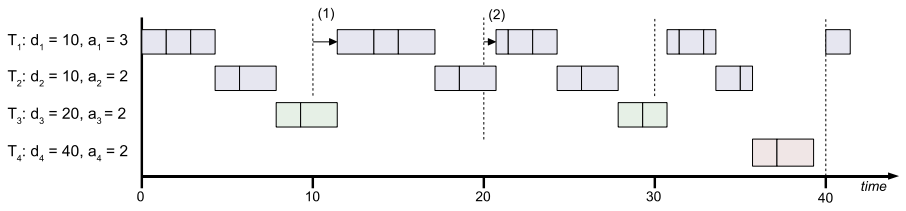


Figure 3.4: Request execution with harmonic periods.

As shown at point (1) in Figure 3.4, the execution of a request of a data stream with a lower priority can overlap with the begin of a new period of a data stream with a higher priority, causing a delay in the execution of the requests of the stream with the higher priority. The maximum delay caused by this overlapping is the worst-case execution time of a disk request. This case is identical to the blocking caused by the overlapping of a disk request at the end of a period as described earlier for uniform periods (illustrated at point (2) in Figure 3.4), and is handled again by including the maximum blocking time in the admission control of the higher priority data streams using $d'_i = d_i - w_{max}$.

Based on the execution scheme shown in Figure 3.4, the admission control can admit the tasks with the shortest period based on the model described for uniform periods. To admit the remaining data streams, the admission control must check whether the execution of the data streams with the higher priorities leaves sufficient time such that the data streams with a lower priority achieve their quality.

To admit a mandatory data stream T_i with a longer period, the aggregated worst-case execution times of all requests with a shorter period $T_1, \ldots, T_{i-1}$ must leave enough time to execute all a_i requests of T_i:

$$\frac{d_i}{d_{i-1}} \left\lceil \frac{d'_{i-1} - w_{d_{i-1}}}{w_i} \right\rceil \geq a_i \tag{3.24}$$

$\left\lceil \frac{d'_{i-1} - w_{d_{i-1}}}{w_i} \right\rceil$ denotes the number of requests of T_i that can be executed using the worst-case execution time as illustrated by Figure 3.5. $w_{d_{i-1}}$ denotes the aggregated worst-case execution time of the data streams $T_1, \ldots, T_{i-1}$, the worst-case execution time of a data stream is defined by

- $w_i \cdot a_i$ for mandatory data streams, and
- $r_i + w_i$ for optional data streams.

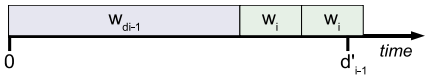


Figure 3.5: Admission control for mandatory data streams with harmonic periods.

To calculate the reservation time r_i for an optional data stream T_i with a longer period, one needs to determine a random variable that describes the portion of the period of T_i that is consumed by the tasks with a shorter period.

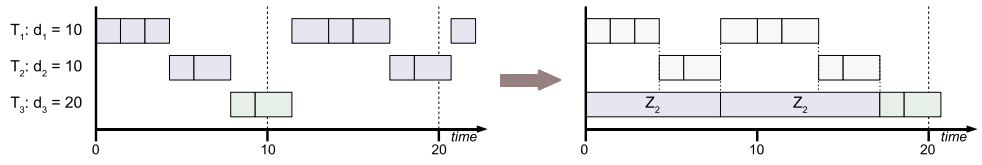


Figure 3.6: Admission control for optional data streams with harmonic periods.

Figure 3.6 illustrates the approach to calculate this random variable Z'_{i-1}. The execution scheme of a task set is transformed such that all requests of tasks with a shorter period than the period of T_i are

executed at the beginning of the period d_i. The number of the shorter periods that fit into the period d_i is defined by $\frac{d_i}{d_{i-1}}$, thus, Z'_{i-1} is given by

$$Z'_{i-1} = \left(\frac{d_i}{d_{i-1}}\right) Z_{i-1} \tag{3.25}$$

where Z_{i-1} is the random variable describing the aggregated execution time of the tasks $T_1, \ldots, T_{i-1}$ for one period d_{i-1}, as defined for the admission control for uniform periods. Using Z'_{i-1} to describe the execution time of the preceding streams, the reservation time for the task T_i can now be calculated as defined by Equation 3.21.

To summarize, the admission control needs to apply the following two steps each time it reaches a new period length:

- Calculate the aggregated worst-case execution time $w_{d_{i-1}}$ of all previous data streams and admit the mandatory data streams with the new period length based on Equation 3.24.
- Determine the random variable Z'_{i-1} to calculate the reservation times for the optional data streams with the new period length.

Operations on Random Variables

The applicability of the admission model described in this section largely depends on the ability to calculate the various operations on random variables.

The admission control uses discrete probability density functions (*p.d.f.*) $f_X(x), x \geq 0$ obtained by measurements to describe the random variables. To calculate the reservation times, the following operations must be performed using these discrete probability density functions:

- The convolution of two probability density functions $f_X * f_Y$. For nonnegative random variables X and Y described by discrete probability density functions f_X and f_Y, the convolution is defined as

$$f_{X+Y}(x) = (f_X * f_Y)(x) = \sum_{i=0}^{x} f_X(i) \cdot f_Y(x-i). \tag{3.26}$$

- The calculation of the probability $\mathbf{P}(A_i(r) \geq k)$, defined by Equation 3.23. Figure 3.7 illustrates the approach to calculate the probability, which is defined by

$$\mathbf{P}(A_i(r) \geq k) = \mathbf{P}(Z_{i-1} + X'_i < d' \ \wedge \ X'_i < r), \qquad \text{with } X'_i = (k-1)X_i = \sum_{j=1}^{k-1} X_i \tag{3.27}$$

  The shaded area highlights the values for Z_{i-1} and X'_i where the condition $Z_{i-1} + X'_i < d \wedge X'_i < r$ is fulfilled. Thus, the probability $\mathbf{P}(A_i(r) \geq k)$ can be calculated as follows:

$$\mathbf{P}(A_i(r) \geq k) = \sum_{z=0}^{n_Z} \sum_{x=0}^{n_X} \begin{cases} f_{Z_{i-1}}(z) \cdot f_{X'_i}(x) & \text{if } (z+x) < d' \ \wedge \ x < r \\ 0 & \text{otherwise} \end{cases} \tag{3.28}$$

  where n_Z and n_X denote the largest elements contained in the random variables Z_{i-1} and X'_i.

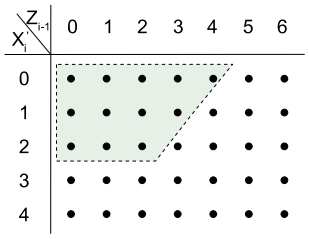


Figure 3.7: Calculation of the probability $\mathbf{P}(A_i(r) \geq k) = \mathbf{P}(Z_{i-1} + X'_i < d' \ \wedge \ X'_i < r)$ with $d' = 5$ and $r = 3$. The shaded area highlights the points where the condition $Z_{i-1} + X'_i < d' \ \wedge \ X'_i < r$ is fulfilled.

- The calculation of the random variable Z_i describing the aggregated execution time of the data streams $T_1, \ldots, T_i$. For a mandatory data stream, Z_i is defined by

$$Z_i = Z_{i-1} + \sum_{k=1}^{a_i} X_i, \tag{3.29}$$

meaning the summation of the execution times of all requests of the data stream with the aggregated execution time of the previous streams.

With optional data streams, not all requests are necessarily executed, the execution stops if either the stream exceeds its reservation time r_i or the execution reaches the end of the period (Eqn. 3.20). To sufficiently consider this behavior in the calculation of Z_i, first the individual execution times $U_{ij}(r_i, d')$ are calculated that describe the execution times of the stream T_i under the condition that exactly j requests are executed. Z_i then results from combining these individual random variables, weighted by the probabilities $\mathbf{P}(A_i(r_i) = j)$ that j requests are executed by the stream:

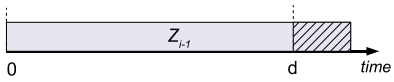
$$f_{Z_i}(z) = \sum_{j=0}^{a_i} \mathbf{P}(A_i(r_i) = j) \cdot f_{U_{ij}}(z) \tag{3.30}$$

The calculation of $U_{ij}(r_i, d')$ requires the exact definition of the conditions under that exactly a given number of requests are executed. Figure 3.8 illustrates the various cases that need to be distinguished to calculate $U_{ij}(r_i, d')$. For each of the cases, the resulting execution time is calculated by a conditional summation of the random variables describing the aggregated execution time of the preceding streams (Z_{i-1}) and of the random variable describing the execution of individual requests of the current stream (X_i). This conditional summation first results in an intermediate random variable $U'_{ij}(r_i, d')$ that needs to be normalized to form $U_{ij}(r_i, d')$. $U'_{ij}(r_i, d')$ is calculated as follows:

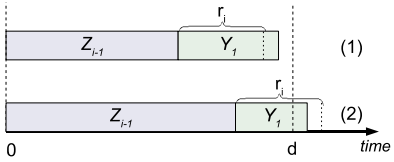
$j = 0$, no request of T_i is executed (Fig. 3.8(a)). This happens if the execution of the data streams $T_1, \ldots, T_{i-1}$, described by the random variable Z_{i-1}, already exhausts the period. U'_{i0} is defined by the part of Z_{i-1} that exceeds the period:

$$f_{U'_{i0}}(u) = \begin{cases} f_{Z_{i-1}}(u) & \text{if } u \geq d' \\ 0 & \text{otherwise} \end{cases} \tag{3.31}$$

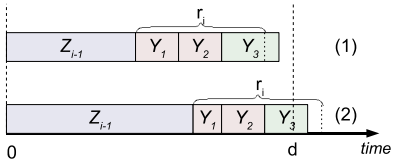
$j = 1$, only the first request is executed (Fig. 3.8(b)). The execution of the previous tasks $T_1, \ldots, T_{i-1}$ did not exhaust the period, but the execution of the first request either reaches



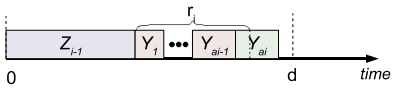
(a) No request is executed for T_i (i.e., $A_i = 0$), the previous tasks exhausted the whole period. U'_{i0} is defined by the shaded part of Z_{i-1}.



(b) To execute exactly one request of T_i, the previous parts must not have exhausted the period, but the execution of the first request of T_i either exhausts the reservation time of T_i (1) or reaches the end of the period (2).



(c) To execute exactly j requests, the execution of the previous $j-1$ requests of T_i must not have exhausted both the period and the reservation time of T_i, but the execution of the j^{th} request either exhausts the reservation time (1) or reaches the end of the period (2).



(d) To execute all requests of T_i (i.e., $A_i = a_i$), the execution of the first $a_i - 1$ requests must not exhaust both the period and the reservation time of T_i.

Figure 3.8: Calculation of Z_i. The Figures show the various cases that need to be distinguished calculating the random variable U_{ij}.

the end of the period or exhausts the reservation time of T_i. U'_{i1} is defined by the conditional summation of Z_{i-1} and X_i (the random variable that describes the execution time of a disk request of T_i) for the values of Z_{i-1} and X_i where the latter conditions are fulfilled:

$$f_{U'_{i1}}(u) = \sum_{z=0}^{u} \begin{cases} f_{Z_{i-1}}(z) \cdot f_{X_i}(u-z) & \text{if } z < d' \ \wedge \ (u \geq d' \ \vee \ u - z \geq r_i) \\ 0 & \text{otherwise} \end{cases} \tag{3.32}$$

$j = 2, \ldots, a_i - 1$, (Fig. 3.8(c)). To execute exactly j requests, the execution of the previous $j-1$ must not have exhausted either the reservation time of T_i or the period, but the execution of the j^{th} request exceeds either of them. Similar to $j = 1$, the random variable U'_{ij} is defined by the summation of the random variable Z_{i-1}, the random variable X'_{j-1} that describes the execution of the first $j-1$ requests of the data stream and X_i for the values of the random variables where the previous conditions are fulfilled:

$$f_{U'_{ij}}(u) = \sum_{z=0}^{u} \sum_{x=0}^{u-z} \begin{cases} f_{Z_{i-1}}(z) \cdot f_{X'_{j-1}}(x) * f_{X_i}(u-z-x) & \text{if } z + x < d' \ \wedge \ x < r_i \ \wedge \\ & \quad \wedge \ (u \geq d' \ \vee \ u - z \geq r_i) \\ 0 & \text{otherwise} \end{cases} \tag{3.33}$$

with

$$X'_{j-1} = (j-1)X_i = \sum_{k=1}^{j-1} X_i$$

$j = a_i$, all requests of T_i are executed (Fig. 3.8(d)), it requires that the execution of all $a_i - 1$ requests did not reach the end of the period and did not exhaust the reservation time of T_i. U'_{ia_i} is defined by:

$$f_{U'_{ia_i}}(u) = \sum_{z=0}^{u} \sum_{x=0}^{u-z} \begin{cases} f_{Z_{i-1}}(z) \cdot f_{X'_{a_i-1}}(x) \cdot f_{X_i}(u-z-x) & \text{if } z + x < d' \ \wedge \ x < r_i \\ 0 & \text{otherwise} \end{cases} \tag{3.34}$$

with

$$X'_{a_i-1} = (a_i - 1)X_i = \sum_{k=1}^{a_i-1} X_i$$

The normalization of the intermediate random variables $U'_{ij}(r_i, d')$ is required to obtain a valid random variable (i.e., to make sure that $\sum_{u=0}^{n_U} f_{U_{ij}}(u) = 1$):

$$f_{U_{ij}}(u) = \frac{1}{\sum_{v=0}^{n_{U'}} f_{U'_{ij}}(v)} \cdot f_{U'_{ij}}(u) \tag{3.35}$$

The aggregated execution time Z_i can now be calculated solving Equation 3.30.

With the described methods to calculate the required operations on the random variables, the admission control is able to solve the admission criteria for optional data streams defined by Equation 3.21.

3.2.3 Disk Model

The admission control depends on the knowledge of both the worst-case execution time w of a disk request and the random variable X describing the execution time of a disk request. Both w and X are parameters of the disk drive and both parameters are presumed to be fixed over time[2]. Thus, these parameters must be acquired for each individual disk prior to its use in the system.

Worst-Case Execution Time w

The maximum time required to execute a disk request is commonly derived using models that reflect the processing of the request by the disk [69, 79, 94]. Figure 3.9 shows the principle scheme of the execution of a disk request the is used to derive the worst-case execution time.

The scheme divides the execution of a disk request into several phases:

- a command processing overhead caused by the host controller and the disk drive,
- a seek time required to move the disk head from its starting position to the track of the target sectors,
- a time to wait until the target sector arrives at the disk head (rotational delay),

[2]Consequently, the dynamic remapping of defect sectors is not considered. However, disk drives allow to monitor this remappings (e.g., using the *Mode Pages* of SCSI disks). This allows the adjustment of the parameters or the replacement of the faulty disk.

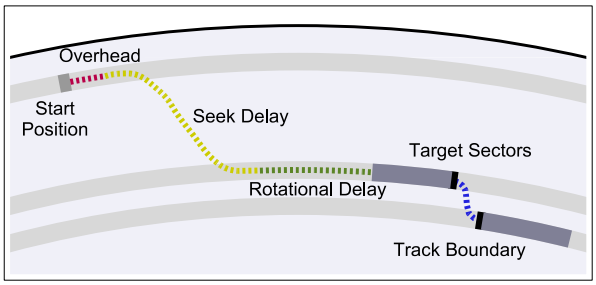


Figure 3.9: Request Execution Model

- the time to access the target sectors, and
- additional track switches if the target sectors of the request span over several tracks.

Based on the described scheme, the following equation defines the worst-case execution time of a disk request [62]:

$$w = t_{maxseek} + n \cdot t_{rot} + m \cdot t_{sector} + v \cdot t_{skew} + t_{ovh} \tag{3.36}$$

with

$t_{maxseek}$ maximum seek time; it is the time for moving the disk head from the innermost to the outermost cylinder of the disk.

$n \cdot t_{rot}$ maximum rotational delay; t_{rot} is the time for a single revolution of the disk, n is the maximum number of revolutions. These additional revolutions are required if the disk fails to settle on the target track.

$m \cdot t_{sector}$ time to access the data; t_{sector} is the time to read / write a single sector on the disk, m the number of sectors to access for a disk request:

$$m = \left\lceil \frac{b}{b_{sector}} \right\rceil \tag{3.37}$$

with b the size of the request and b_{sector} the size of a sector on the disk.

$v \cdot t_{skew}$ time to switch to the next track or head; t_{skew} is the time for a single switch, v the maximum number of switches which can occur executing a disk request. v depends on the size b of a disk request and the minimum size of a disk track b_{track}:

$$v = \left\lceil \frac{b - b_{sector}}{b_{track}} \right\rceil \tag{3.38}$$

t_{ovh} request processing overhead.

The calculation of both m and v depend on the request size, thus, the admission control must use a worst-case execution time w_i that corresponds to the block size b_i used by the data stream.

Request Execution Time X

The admission control uses the random variables X_i to describe the actual distribution of the execution times of a disk request. In contrast to the worst-case execution time, the random variables X_i are obtained by direct measurements, creating a distribution of the relative frequencies of the execution times.

To achieve a high accuracy with the admission control, the workload used to measure the random variables X_i should resemble the expected workload of the real system. This match between the workloads can be achieved by either

- replaying traces of real workloads, or
- create synthetic workloads that reproduce the behavior of a real system.

Two properties of the task model used by the admission control (Eqn. 3.10) allow to further improve the accuracy of the random variables. First, data streams are either read from the disk or written to the disk, and second data streams are read or written using a fixed request size. These properties can be exploited by obtaining several distributions, distinguishing between different request sizes as well as between read and write requests. The admission control than can choose the appropriate distributions based on the properties of the streams.

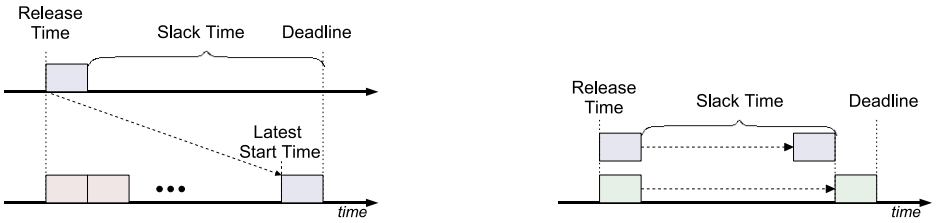
3.3 Disk-Request Scheduling

The admission model presented in the previous section calculates a reservation time for each stream such that the stream achieves its requested quality (the reservation times for mandatory streams are set to the worst-case execution time of their requests). The task of the disk-request scheduler is to bring in line the enforcement of these reservation times with the optimization of the disk utilization as well as with the support for further quality-of-service types, such as bounded response times.

A straightforward approach to enforce the reservation times is to execute the requests of the streams in the order of the priorities assigned by the admission control and limiting the amount of time each stream is able to consume. However, this approach yields two major limitations. First, it limits the ability of the scheduler to optimize the execution order of the requests, the scheduler is only able to reorder the requests of the stream that is currently executed. Second, requests of clients that are not included in the admission control (i.e., the requests of sporadic real-time and best-effort applications) can only be executed once all streams are processed by the scheduler in a period. This moves the execution of these requests towards the end of a period, which can result in the potential miss of a deadline of a sporadic real-time request although the disk would have been able to execute this request earlier, as well as long response times for best-effort requests.

These limitations mainly result from not utilizing the full flexibility provided by the stream model. Although the stream model only requires the execution of the requests to be finished at the end of a period to successfully meet the quality-of-service requirements, executing the streams strictly based on their priorities causes the execution of the requests usually to be finished well ahead of this deadline. Thus, applying the admission priorities to the actual request scheduling is an overly stringent approach. To ensure that all streams achieve their requested quality, it is sufficient to make sure that each stream is able to make use of its reservation time within a period, independent of both the exact time within the period and whether the time is consumed all at once or in smaller units.

The disk-request scheduler can make use of the flexibility provided with the execution of the requests of data streams by using a slack-stealing scheduling policy [46]. The slack time of a disk request is



(a) Without slack stealing, the execution of a real-time request can be finished well ahead of its deadline. This earliness can be utilized to execute other requests ahead of the real-time request.

(b) The calculation of the available slack time must respect the requirements of all real-time request of the system. To ensure that the lower request in the figure is able to meet its deadline, the upper request cannot be delayed until its deadline.

Figure 3.10: Slack-Stealing Disk Scheduling. The principle idea of the slack-stealing disk scheduling is to postpone the execution of a real-time disk request until the latest time the execution needs to be started such that the execution is finished ahead of its deadline. The gained time can be used to execute other request, for instance to optimize the execution order or to provide best-effort requests with short response times. The execution must not be necessarily postponed until the latest execution time, for instance a request scheduler that optimizes the disk utilization can execute real-time requests either if they fit into the optimized execution order or if they must be executed to meet their deadlines.

defined by the time that the execution of the request can be postponed without violating its deadline (illustrated in Fig. 3.10). The time gained by postponing a request can be used to execute other requests, either to provide better response times for these requests or to optimize the overall execution order of the requests. Applied to the enforcement of the reservation times, this means that the execution of the requests of a data stream can be postponed in favor of a lower priority request (including sporadic real-time and best-effort requests) as long as the scheduler makes sure that the stream is able to make use of its reservation time until the end of its period.

The main challenge with a slack-stealing scheduling policy is to calculate the exact amount of time the execution of a request can be postponed. In particular, the calculation has to consider that the execution of a request can be further postponed by the execution of requests with a higher priority, thus making the slack time not only to depend on the distance of the request to its deadline, but also on the amount of time used by higher-prioritized streams. To clearly separate this calculation from the task to find the next request to execute, one can use a two-level approach:

1. Each time the scheduler needs to pick a request for execution, a subset of the outstanding requests is created. The creation starts with an empty subset and adds the requests of the data streams, beginning with the stream with the highest priority, as long there is enough time left in the period such that each stream in the subset is guaranteed to utilize its reservation time (i.e., the slack time of the streams already contained in the subset allows to add further requests). Sporadic real-time requests and requests of best-effort applications are allowed to be added to this subset only if there is sufficient slack time available once all streams are added to the subset.

2. With the reservation times sufficiently enforced by the creation of the subset, a traditional request-scheduling policy can be applied to this subset to find the request that will be executed by the disk.

A main property of this approach is that it is applied *each* time the scheduler needs to choose a request, which provides a greater flexibility in the calculation of the slack times than with the approaches described in Section 2.2 that use precomputed slack times.

The subset is named *Dynamic Active Subset* (DAS) [66], it contains all requests that can be considered by the scheduler at a given time.

3.3.1 DAS—Dynamic Active Subset

The task of the Dynamic Active Subset is to ensure that each stream is able to consume the reservation time calculated by the admission control. The execution of a stream can only be postponed in favor of a lower-priority request if the request scheduler can make sure that *after* the execution of this request still enough time is left such that the stream is able to consume its reserved time. The creation of the DAS ensures this behavior by adding streams to the subset only as long as the slack time provided by the set of streams already included in the subset allows the execution of further requests.

The general approach to calculate the slack time of a set of streams is to derive the difference between the time left until the end of the period and the time required by the streams of that set (i.e., the amount of time the streams still need to consume in the period to meet the reservation time). The following discussion describes the calculation of the slack time and the creation of the subset in detail, again starting with a description for task sets with uniform periods and providing a generalization to harmonic periods afterward.

Uniform Periods

Figure 3.11 depicts the calculation of the slack time for data streams with uniform periods. The time t_{left} denotes the remaining time in the period at the point where the subset is calculated.

(a) Slack time of a mandatory data stream.

(b) Slack time of a mandatory and an optional data stream.

Figure 3.11: Slack time of data streams. t_{left} is the remaining time in the current period.

Figure 3.11(a) illustrates the calculation of the slack time for a mandatory data stream. To ensure that all remaining requests of the data stream are executed prior to the end of the period, the execution must start at least a time $u(t)$ ahead of the end of the period such that all requests can get executed even under worst-case conditions. $u(t)$ denotes the time the mandatory stream still needs to be able to consume within the period and it depends on the number of outstanding requests for that stream:

$$u(t) = a' \cdot w \tag{3.39}$$

where a' specifies the outstanding requests of the data stream in the current period and w the worst-case execution time of a disk request of the stream. Thus, with a remaining time t_{left} in a period, the slack time of the mandatory data stream shown in Figure 3.11(a) is given by

$$s = t_{left} - a' \cdot w \tag{3.40}$$

For optional data streams, the scheduler must ensure that the streams can fully utilize their reservation times. The scheduler maintains a budget for each data stream, the budget is replenished with the reservation time at the beginning of each period and the actual execution times of the disk requests are accounted to this budget. Because requests of a data stream are executed as long as the budget is not zero, the overall time consumed by a data stream can be more than its reservation time (i.e., the execution of the final request can take more time than it is left in the budget). Thus, the calculation of the slack time for an optional stream considers the remaining budget of the stream (r'_i) and the maximum time the request execution can exceed the budget, which is given by the worst-case execution time w_i. For the example shown in Figure 3.11(b), this results in a slack time defined by

$$s = t_{left} - (\overbrace{a'_1 \cdot w_1}^{(1)} + \overbrace{r'_2 + w_2}^{(2)}) \tag{3.41}$$

Both terms describing the time required to execute the mandatory stream (1) and the optional stream (2) denote the share $u_i(t)$ of the remaining time required to fulfill the quality-of-service guarantees of the streams. With the described approach, one can calculate the combined slack time of data streams $T_1, \ldots, T_i$ as follows:

$$s_i(t) = t_{left} - \sum_{j=1}^{i} u_j(t) \tag{3.42}$$

with:

$$u_j(t) = a_j(t) \cdot w_j \qquad \text{for mandatory data streams} \tag{3.43}$$
$$a_j(t) \text{ is the number of outstanding requests at time } t$$
$$u_j(t) = r_j(t) + w_j \qquad \text{for optional data streams} \tag{3.44}$$
$$r_j(t) \text{ is the remaining budget of the stream at time } t$$

Given this method to calculate the slack time, the creation of the DAS adds streams to the subset as long as the slack time of the streams already in the DAS is large enough to postpone the execution of the streams and the budget of an optional stream is not exceeded. Thus, the following condition defines the criteria to add a stream to the DAS:

$$s_{i-1}(t) > w \ \ \wedge \ r_i(t) > 0 \tag{3.45}$$
$$s_0(t) = t_{left}$$

The condition ensures that a stream is added to the DAS only if it can be ensured that the streams already contained in the DAS can still get their time u_i assigned to achieve their quality-of service guarantees, even if the scheduling policy picks a request of the newly added stream, as illustrated in Figure 3.12.

It is important to note that if Condition 3.45 holds for a data stream, *all* requests of the stream are added to the DAS. This maximizes the number of requests from which the scheduling policy can pick.

Harmonic Periods

The calculation of the slack time of a task set with harmonic periods needs not only to consider the time $u_i(t)$ required by a stream within the current period. It must also include the case that streams with a shorter period execute the requests of several periods within a single period of a stream with

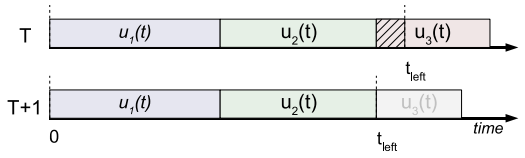


Figure 3.12: Creation of the DAS. At time T, Condition 3.45 holds for all three data streams, meaning that all three streams can be added to the DAS. The request scheduling policy picks a request of the third stream for execution, the execution time is denoted by the shaded area. Thus, at time $T+1$ the Condition 3.45 holds only for the first two data streams, meaning that the DAS at time $T+1$ only consists of the requests of stream one and two. This ensures that the first two streams can achieve their quality-of-service guarantees.

a longer period. Figure 3.13 illustrates such a case. The slack time s_1 for task T_1 at point (1) is calculated as described for uniform periods. To calculate the slack time s_2 of task T_2 at point (1), one does not only need to consider the utilization of task T_1 in its current period, but also that the whole task T_1 is executed two additional times until the end of the period of T_2. This further reduces the time available for the execution of requests of task T_2.

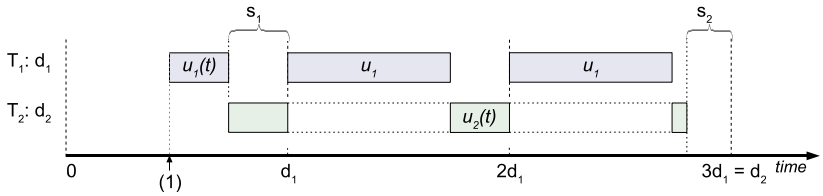


Figure 3.13: Creation of the DAS for harmonic periods. The calculation of the slack time s_2 at point (1) must consider the time required by the stream T_1 both in the current period of T_1 (denoted by $u_1(t)$) and for all further periods of T_1 that occur until the end of the period of T_2 (u_1 denotes the time required by T_1 in a full period).

To sufficiently include this behavior in the calculation of the slack time of T_2, the time $t_{left,2}$ that denotes the available time until the end of the period of T_2 is only allowed to include the time not consumed by T_1 in that time span. At point (1), it is given by:

$$t_{left,2} = \overbrace{t_{left,1} - u_1(t)}^{(a)} + 2 \cdot \overbrace{(d_1 - u_1)}^{(b)} \tag{3.46}$$

The first part of the equation denotes the time not consumed by T_1 in the current period of T_1 (i.e., the slack time s_1 of T_1 in this period). Part (b) denotes the overall time left over by the execution of the requests of T_1 in a period. u_1 denotes the maximum *utilization* that can be caused by the execution of the stream. In general, it is defined by:

$$u_i = a_i \cdot w_i \qquad \text{for mandatory data streams, and} \tag{3.47}$$

$$u_i = r_i + w_i \qquad \text{for optional data streams.} \tag{3.48}$$

The slack time s_2 is then given by the difference between $t_{left,2}$ and the time required by T_2 in the current period:

$$s_2 = t_{left,2} - u_2(t) \tag{3.49}$$

To generalize this approach, one needs to be able to recalculate the remaining time t_{left} in a period each time the creation of the DAS reaches a stream with a longer period. A generic definition of the calculation of this time is given by:

$$t_{left,i+1} = s_i + n \cdot (d_i - u_i) \tag{3.50}$$

n denotes the number of complete periods with the shorter period length d_i that occur until the end of the new, longer period. u_i denotes the maximum amount of time the execution of *all* requests of the streams $T_1, \ldots, T_i$ requires in a period d_i.

The slack time of a task T_i is then defined by

$$s_i = t_{left,i} - u_i(t) \qquad i = 1, \ldots, n \tag{3.51}$$

and $t_{left,i}$ is given by

$$t_{left,i+1} = \begin{cases} t_{left,i} - u_i(t) & \text{if } d_i = d_{i+1} \\ s_i + n \cdot (d_i - u_i) & \text{if } d_i < d_{i+1} \end{cases} \tag{3.52}$$

The DAS can now be created as described for uniform periods, adding streams to the subset as long as Condition 3.45 holds.

Acceptance Test for Sporadic / Aperiodic Tasks and Best-Effort Requests

Requests of clients not included by the admission control, such as requests of sporadic and aperiodic tasks that are not mapped to a data stream or requests of best-effort clients, can only be added to the DAS if sufficient slack time is available after adding the last stream (i.e. $s_n \geq w$).

To ensure that requests of sporadic and aperiodic tasks meet their deadline once they are admitted to the system, additional requests of best-effort clients are again only allowed to be added if sufficient slack time is available after the former requests are added to the DAS. The slack time of a request with a deadline is calculated as shown in Figure 3.10.

Optional Streams that Exhausted their Reservation Time

Based on Condition 3.45, the creation of the DAS includes requests of optional data stream only as long as the streams did not exceed their budget for the current period. The system is free to add remaining requests of streams that exceeded their budget to the DAS together with best-effort requests if the slack time allows this, or the system can discard these requests completely to increase the bandwidth available for best-effort requests.

3.3.2 Request Scheduling

The purpose of the DAS is to release the request-scheduling policy from taking care of the quality-of-service guarantees. The system can choose a specific policy to pick a request from the subset depending on further requirements of the system. For instance, to optimize the overall disk throughput a policy such as *Shortest Access Time First* (SATF) [42], *Shortest Seek Time First* (SSTF) [74], or SCAN [74] can be used. Other policies are possible to provide bounded response times for best-effort requests, for instance *Shortest Access Time First with Urgent Forcing* (SATFUF) [42]. By prioritizing

best-effort request over requests of data streams a schedule can be created similar to the resulting schedule of the ΔL scheduler [11].

The only constraint a scheduler needs to take care of is the dynamic creation of the request subset. The DAS is created each time a request needs to be picked for execution, and due to the changing slack times the number of streams that are added to the DAS can change. In particular, data streams with a low priority and best-effort requests might be added only occasionally to the subset. This avoids the use of globally optimizing scheduling policies such as *Optimal Access Time* (OAT) [42].

3.4 Summary

This chapter presented the Quality-Assuring Disk Scheduling model and a disk-request scheduler based on the Dynamic Active Subset.

The Quality-Assuring Disk Scheduling provides an admission model that incorporates both hard real-time guarantees and statistical real-time guarantees. The admission control is based on the calculation of a reservation time for each stream such that the stream achieves its quality-of-service requirements.

The disk-request scheduler enforces the reservation times using a two-level approach. First, the scheduler creates a subset of the outstanding requests, trying to include as many requests as possible into this subset, but always ensuring that each stream is able to utilize at least its reservation time. Then, a request scheduling policy is applied to this subset to decide which of the requests is executed. This two-level approach clearly separates the enforcement of the real-time guarantees (i.e., of the reservation times) from the actual request scheduling.

The following chapter will discuss how a file system can incorporate the requirements posed by the described models.

Chapter 4

File Systems with Quality-of-Service Guarantees

The previous chapter presented an admission model and a disk-scheduling algorithm that are both capable of providing quality-of-service guarantees. Based on the requirements raised by the admission model, the following chapter will describe the design of a file system that provides quality-of-service guarantees to files.

The main parts of a file system implementation are [26, 85]:

Disk Space Management The main task of the disk space management is to provide an allocation mechanism deployed by the file system to allocate the disk blocks used to store the file data.

File Representation The file system needs to keep track of which blocks on the disk belong to a file.

Client API The clients of a file system require an interface to access and modify the files of the file system. This particularly includes a naming scheme.

File System Performance File systems can severely affect the performance of the overall system, this requires the deployment of optimization strategies to improve the performance of the file system, such as throughput, latency or number of file operations per second.

File System Reliability and Security File systems can include access control mechanisms and precautions to improve the robustness of the file system.

Each file system has to address these tasks in the context of the target system. For the scope of this thesis, this context is defined by:

- The disk admission model; the main properties of the model described in the previous chapter are
  - Disk requests of data streams are periodic.
  - The admission model requires that files storing the data of real-time streams can be accessed with a fixed request size. The required request size can vary for each stream of the file system.
- The DROPS application model; real-time applications consist of a chain of components processing a data stream contiguously. Additionally, the file system should support traditional, non–real-time applications.

The following discussion will present the design of a file system that considers these requirements. It will focus on the first three tasks of a file system. The DAS scheduler already addresses the performance optimization, additional optimizations such as the use of caches will be discussed as part of the client API. The reliability and security of file systems, for instance the use of journaling or encryption, are beyond the scope of this work.

4.1 Block Allocation

To comply with the data stream model presented in Section 3.1, the file system has to ensure that files can be accessed with the fixed disk-request size b. To ensure this request size, the files must be stored in contiguous chunks of size b or a multiple thereof. The required request size varies for each data stream, video streams require a larger request size to achieve their bandwidth than streams with a lower bandwidth, such as audio streams.

A widely used approach to allocate disk space is the allocation of fixed-sized disk blocks [53, 85], mostly implemented using bitmaps to keep track of the allocated and free disk blocks. To ensure the minimum request size for all data streams, the block size would need to match the request size of any stream stored by the file system, which can be in the order of 64 KByte to 256 KByte for video streams. However, such a large block size wastes disk space for smaller files (the so called *Internal Fragmentation* [93]), especially for the files of a time-sharing system, which have an average size of 16 KByte to 128 KByte [59, 20]. Thus, the allocation policy should support several block sizes, allowing the use of an appropriate block size for each file individually. Several file systems attempt to provide such a flexible allocation policy, aiming mainly at the improvement of the file system performance by enabling larger disk transfers:

- *Clustering* of smaller disk blocks in the UNIX file system [55]; the file system allocates several contiguous disk blocks to provide a larger allocation size. However, the file system does not guarantee a cluster size, an allocation request might be split into several smaller clusters. The allocation of a specific cluster size would require an expensive search in the allocation bitmap to find an appropriate set of contiguous blocks and is vulnerable to external fragmentation.
- Extent-based allocation using trees, such as in the XFS [83] and ReiserFS [65] file systems. The file system maintains trees storing information about the free extents on the file system. Although trees allow the fast lookup of free extents, the maintenance of the trees is more complex than a bitmap-based allocator (e.g., XFS maintains two trees to efficiently support all required lookups).
- Block allocation based on the *Buddy* allocator [45]. It requires an expensive reallocation algorithm to eliminate the external fragmentation caused by the buddy allocator.

To comply with the data stream model, an actual system does not need to provide an allocation policy that supports arbitrary block sizes. Depending on the work load, only a view block sizes will be used, for instance a large block size (in the order of 64 KByte–256 KByte) to store high-bandwidth video streams, a medium block size (16 KByte–32 KByte) to store low-bandwidth audio streams and a small block size (4 KByte) to store the files of the best-effort system. This significantly reduces the required flexibility of the allocation policy.

Figure 4.1 illustrates the approach for such a policy. The available disk space is divided into several allocation groups, similar to the cylinder or block groups in the UNIX file system. Each allocation group stores the file data using a fixed block size, but the block size can vary between the allocation groups.

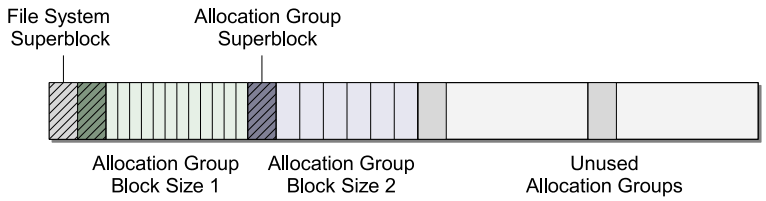


Figure 4.1: File System Block Allocation. The available disk space is divided into allocation groups, each allocation group provides a fixed block size. The assignment of the block sizes to the allocation groups is done dynamically, which allows the allocation policy to adapt to changing workloads.

The file system dynamically assigns block sizes to allocation groups, with groups initially marked unused. The file system allocates a new allocation group for a block size if no more free blocks of that size are available in other groups. If all blocks of an allocation group are freed, the group is released and can be reused for other block sizes. The implementation is similar to the UNIX file system, the header of each allocation group contains a bitmap storing the information about the available blocks in that group. Additionally, the file system superblock contains a bitmap maintaining the information about the available allocation groups.

This approach provides several advantages:

- It is expected to perform comparable to the UNIX allocation policy. Only a small overhead is caused by the additional management of the allocation groups.
- Because the block size used for a file is fixed, the blocks of a file are allocated from the same allocation group, or if not large enough from only a few groups, providing allocation locality. Additionally, well-known approaches to further improve the allocation locality can be applied, such as the allocation of the file blocks in the same allocation group as the file's I-node.
- The fragmentation of the free space is limited to the free space assigned to allocation groups with different block sizes.
- The approach is an *integrated design* [78], providing a flexible allocation of disk space to the various file types. This allows a dynamic adaptation to changing workloads.

4.2 File-System Metadata

In addition to the file data, a file system needs to store management data, the file system *metadata*. The two main types of metadata are first the block allocation bitmaps described in the previous section, and second the data structures used to keep track which disk blocks belong to a file.

4.2.1 File Representation

With the allocation policy described in the previous section, the file data is stored in a set of equal-sized disk blocks. The file system needs to maintain a list of blocks that belong to each file. The use of fixed-size blocks allows the implementation of a direct-indexed block list, such as the I-node structure used by the UNIX file system (Fig. 4.2). A tree-based file implementation, as used by XFS or ReiserFS, is of no advantage. It is suitable with an allocation policy based on variable-sized extents.

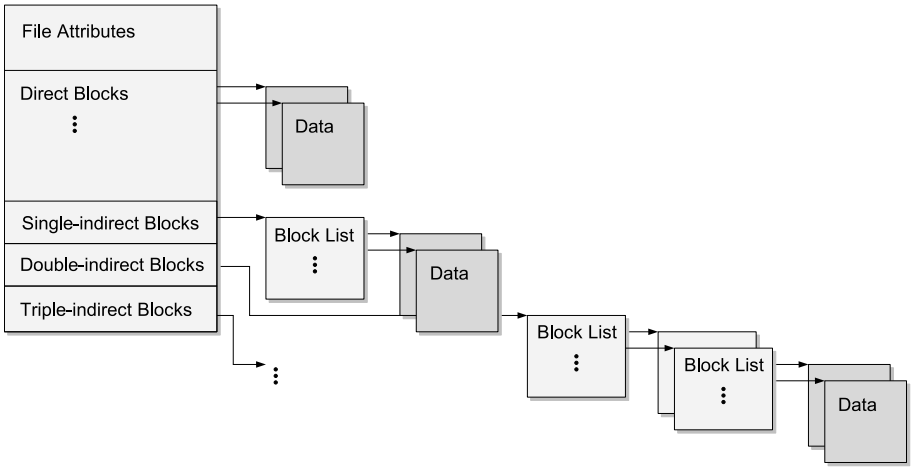


Figure 4.2: Inode Structure.

Similar to the UNIX file system, the superblocks of the allocation groups store I-node tables, and the disk blocks storing the pointers to indirect blocks are preferably allocated from the I-node's block group.

4.2.2 Scheduling of Metadata Requests

To ensure the timely processing of the disk requests, the file system needs to have all metadata available required to create the disk requests prior to the request's period. If additional disk requests are necessary to obtain this metadata, these requests need to be scheduled with time constraints. The important information is

- the list of block numbers to generate the requests;
- the allocation bitmaps to allocate blocks for recorded streams if the block allocation is done dynamically.

Other metadata, such as directories, do not affect the creation of the stream disk requests, thus need no special attention.

Block Lists

To obtain the block list of a file, the file systems needs to read the corresponding parts of the file's I-node structure. To read a block-list block with an I-node structure as shown in Figure 4.2, the file system might need to read additional blocks storing the indirect block lists. These additional requests need to be considered in the scheduling of the metadata requests.

The timely availability of the block list can be ensured by several approaches:

- Interlace the requests to read the block list with the request stream of the file, and increase the bandwidth requirement of the stream appropriately. However, this only works with mandatory data streams, where the processing of all requests is guaranteed.

Block Size b_{ag} (KByte)	4	8	16	32	64	128
Number of Blocks $\left(n_{blocks} = \frac{b_{ag}}{4\,\text{Bytes}/\text{Block}}\right)$	1024	2048	4096	8192	16384	32768
File Size (MByte)	4	16	64	256	1024	4096

Table 4.1: Number of block numbers stored in one disk block of the block list and the file size covered by these blocks depending on the block size. The block numbers are 32 Bit values.

- Process the metadata requests in a separate, mandatory data stream. Because the bandwidth of such a metadata stream is extremely low for a single file (in the order of 1 request per 30 minutes for video streams), the metadata requests of several streams can be multiplexed to a single stream.
- Read the full request list prior to the start of the stream.

With the block-list blocks ordinarily allocated from the allocation groups, the number n_{blocks} of block numbers stored in a single disk block of the block list depends on the block size b_{ag} of the allocation group. Table 4.1 shows the number of blocks and the corresponding file size depending on the block size. The numbers show that only a few blocks are required to hold the block list of a file if an appropriate block size is used. For instance, a one-hour MPEG-2 video (4 MBit/s, 1.75 GByte) requires 28800 blocks if stored in 64 KByte blocks, which can be held in a block list of just two disk blocks. Thus, despite an I-node structure using indirections, the file system is able to read the disk blocks storing the block lists without additional requests to read intermediate block lists. The few numbers of block-list blocks can be held in memory, reading the intermediate block list (i.e., the first level of the double-indirect block list) prior to the processing of the stream. The memory requirement is moderate, the block list of the MPEG-2 example requires 112.5 KByte of memory.

Allocation Bitmaps

To write a data stream, the file system needs to allocate the required disk blocks prior to the time they are used to store the data of the stream. The number of disk requests required to allocate a set of disk blocks can significantly vary depending on the fragmentation of the file system. In the best case, all blocks are allocated from the same allocation group, requiring just one disk request to access the allocation bitmap of that group. In the worst case, each block is allocated from a different allocation group, requiring an additional disk request for each block to allocate. Additional disk request might be necessary to allocate and initialize new allocation groups.

Due to the varying number of disk requests required to allocate the disk blocks, an online allocation of the disk blocks during the recording of the data stream is not feasible. Instead, the required blocks to record the data stream are allocated prior to the start of the recording, the number of blocks can be calculated from the duration and the bandwidth of the data stream. This preallocation also ensures that sufficient space is available to store the data stream. Thus, it has to be done as part of the admission control for a stream written by the file system.

4.3 Client Interface

The client interface of a file system that supports various file types has to address the varying characteristics of these file types. Based on the admission model, the file system needs to support the following file-request types:

- contiguous data streams,
- requests with individual deadlines, and
- requests of non–real-time applications.

4.3.1 Contiguous Data Streams

The data-stream model defined in Section 3.1 raises various requirements on the way the file system generates the disk requests. The main characteristics of the stream model are (illustrated in Figure 4.3):

- The deadline for all requests of a period is the end of the period. The order of execution of the requests within a period is not predictable, it is dynamically determined by the DAS scheduler.
- The latest release time for all requests of a period is the begin of that period.

With these requirements, a traditional synchronous `read()`/`write()` client interface is not applicable to the file system. Instead, the file system must be able to generate the disk requests for the data streams independently of the retrieval or delivery of the data by the clients. The file system could achieve this by providing an asynchronous client interface, requiring the applications to submit their requests ahead of the actual retrieval of the data. However, such an asynchronous interface would require major changes to the applications, which limits the practicability of this approach.

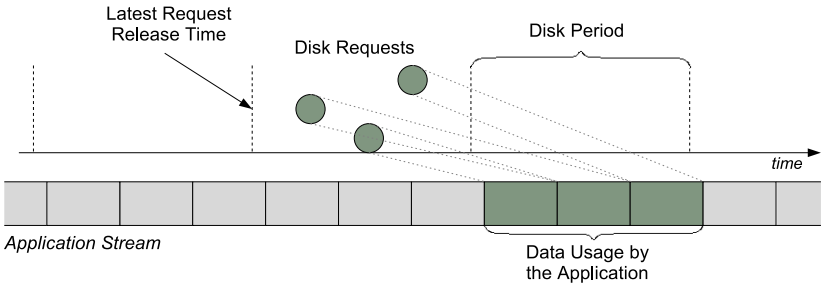


Figure 4.3: Request Generation for Data Streams. The figure denotes the constraints the file system needs to consider with the generation of the disk requests to timely deliver a stream to an application. The deadline for the execution of the disk requests is given by the end of a period used by the disk scheduler to process the requests. The file system must generate the disk requests such that they are executed at the latest in the period that ends just before the application retrieves the data. This sets the latest time the file system must generate the requests to the beginning of that period.

Another, more practicable approach is the use of prefetching (to read a data stream) and buffering (to write a data stream). With the contiguous processing of the data streams, the file system is able to generate the requests by itself, without the assistance of the clients. This way, the file system can generate the requests such that the data is available upon retrieval of a read stream by the client, and that the data is timely written for write streams.

The prefetching and buffering approach requires the allocation of sufficient buffer memory by the file system to handle the different characteristics of read and write data streams.

Read Data Streams

For data streams read from the disk and delivered to an application, the required amount of buffer is determined by two factors (illustrated in Figure 4.4):

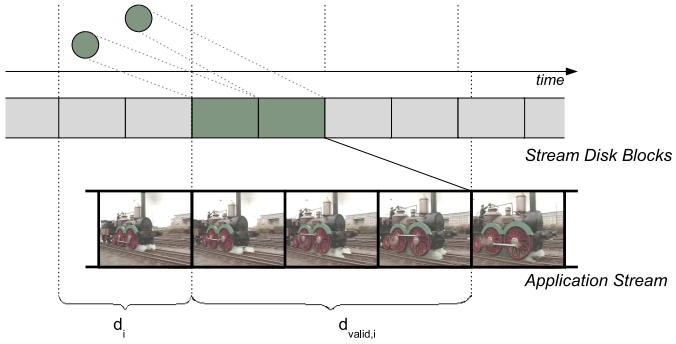


Figure 4.4: Buffer Requirements of Read Streams. The file system must allocate a sufficient amount of memory to both hold the data processed within a disk-scheduling period and to make this data available to the application for the whole time span the application needs to access the data. The latter time span depends entirely on the application and particularly can be much longer than a disk-scheduling period.

1. the length d_i of the disk period; the buffer memory must be available at the release time of the requests.
2. the time d_{valid_i} the client requires the availability of the data, starting at the end of the related disk period. This time solely depends on the characteristic of the client application, in particular it can exceed a disk period.

Thus, the minimum time a buffer element needs to be available for a data stream T_i is

$$d_{buffer,min_i} = d_i + d_{valid_i}$$

and this results in a minimum buffer size[1] of

$$s_{buffer,min_i} = (a_i \cdot b_i) \left\lceil \frac{d_i + d_{valid_i}}{d_i} \right\rceil \tag{4.1}$$

where a_i denotes the number of disk requests processed by the disk scheduling for the stream in a single period and b_i the request size used by the stream (thus, $a_i \cdot b_i$ is the amount of data read in a single disk period). $\left\lceil \frac{d_i + d_{valid_i}}{d_i} \right\rceil$ denotes the number of disk periods covered by the time a buffer element needs to be available.

Write Data Streams

The memory required to buffer the data of a stream written to the disk depends on:

1. the length of the disk period; all requests must be available at the beginning of the period and the buffer memory is released at the end of the period.
2. the amount of memory required to store the data delivered by the client, it depends on the burstiness of the delivery.

The first condition can be handled identically to read streams, the second condition requires knowledge of the behavior of the applications generating the streams. In particular, the file system needs to know the maximum amount of data the application generates within a time interval.

[1] The buffer size does not include the memory required to handle VBR streams as described in Section 3.1.

With the generation of the disk requests entirely handled within the file system, the main task of the client interface is to allow the efficient transfer of the stream data between the file system and the clients. To avoid copy operations between the file system and the clients, the file system can share the stream buffer with the clients, similar to Fbufs [21] or IO-Lite [60]. However, a simple shared-memory buffer is not sufficient to provide the required data-stream semantics, this additionally requires

- an ordering scheme within the buffer, and
- a synchronization mechanism.

The synchronization mechanism has to accomplish several requirements. First and foremost, it has to govern the accesses to the elements of the buffer, making sure that both the receiver of a stream (i.e., the client that retrieves a stream or the file system that writes a stream) only accesses valid data and that the sender of the stream does not overwrite buffer elements not yet processed by the receiver. This requirement is of particular interest to be able to adapt the processing speed of the file system to the behavior of the applications, and in particular to cope with the varying bandwidth requirements of variable-bit-rate streams. To stop the generation of disk requests to read a stream, the file system needs to be able to detect whether the client is able to process the data fast enough or not (i.e., if the buffer is entirely filled with unprocessed data). Similarly, the file system needs to be able to detect whether the buffer contains valid data to write a data stream or if it has to stop the generation of disk requests because the client does not produce the data fast enough. In addition to govern the buffer accesses, the synchronization mechanism must also provide the means to signal disk requests of optional streams discarded by the disk scheduler to the clients.

The data-stream semantics can be achieved by organizing the stream buffer as a synchronized, circular buffer (denoted in Figure 4.5), with each buffer element storing the data to be handled by single a disk request. The circular buffer ensures that the streams are accessed in the right order. Aligning the size of the buffer elements with the request size of the disk stream allows to apply appropriate synchronization primitives on a per-request basis, allowing the file system both to test whether it can create disk requests and to signal discarded requests to its clients.

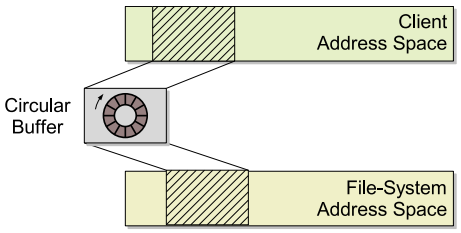


Figure 4.5: Streaming Client Interface. The streaming interface uses a circular buffer that is shared between the file system and the client. The circular buffer both provides synchronization mechanisms and an ordering scheme for the stream data.

To summarize, the file system provides a streaming interface to support contiguous data streams. The data is transfered between the file system and the clients using a circular buffer implemented on memory shared between the file system an the client, providing an `mmap()`-like file access in the clients. The interface provides operations to create and destroy a stream, to start, stop and resume the processing of the stream and to position the file pointer within the stream. The creation of the stream includes the admission of the disk stream based on the bandwidth requirement specified by the client.

4.3.2 Requests with Individual Deadlines

As described in Chapter 3, real-time requests not mapped to data streams are not considered by the admission control, they are handled by the DAS scheduler using a per-request acceptance test. Thus, it is sufficient to extend a traditional file system interface with a means of specifying a deadline for each `read()` or `write()` request and an additional error code to signal the client whether the request was admitted by the acceptance test or not.

4.3.3 Non–Real-Time Requests

To support non–real-time applications, the file system should provide an established, POSIX-like interface [41]. The interface provides `read()` and `write()` operations on files using file descriptors as well as memory mapped file accesses.

4.4 Summary

This chapter described the design of a file system that uses the DAS scheduling to provide file accesses with quality-of-service guarantees. The design focuses on the parts of a file system that are influenced by the underlying disk scheduling, namely the block allocation policy, the metadata management and the client interface.

To comply with the requirements of the disk stream model, the block allocation policy presented in Section 4.1 allows the allocation of blocks with several block sizes. The available disk space is divided into several allocation groups, each group offering a fixed block size. The allocation groups provide an efficient allocation mechanism as well as limit the external fragmentation of the disk space.

In Section 4.2 I discussed the structure of the file system metadata and presented approaches to schedule the disk requests necessary to access the metadata of real-time data streams. To access the block list of a file, the required disk requests can be either mapped to a separate data stream or the block list can be stored entirely in memory. To record a data stream, the required disk blocks need to be preallocated prior to the recording of the stream.

Finally, in Section 4.3 I presented a file-system client interface that supports various file types. It offers a streaming interface to support contiguous data streams and a POSIX-like interface to support file accesses with individual deadlines as well as non–real-time file accesses.

Chapter 5

The DROPS Disk-Storage System

The previous two chapters presented the design of a disk subsystem that supports quality-of-service guarantees and the design of a file system that meets the requirements caused by this disk subsystem. The following chapter outlines the implementation of these designs with the DROPS Disk-Storage System, particularly demonstrating the feasibility of the presented concepts in a real-world implementation.

Section 5.1 provides an overview of the implementation, Section 5.2 describes the disk request scheduling based on the DAS and Section 5.3 describes the adaptations applied to an existing file system to provide a block allocation supporting various block sizes and the implementation of the streaming client interface. Finally, Section 5.4 concludes with a summary.

5.1 Overview

The DROPS Disk-Storage System is a part of the *Dresden Real-Time Operating System* (DROPS) [37], which provides an environment for the development of real-time applications, in particular multimedia applications. DROPS is based on the FIASCO [36] microkernel, an implementation of the L4 microkernel API [48] incorporating extensions to support the scheduling of real-time threads [82].

Figure 5.1 illustrates the structure of the DROPS Disk-Storage System. It consists of two main elements, the L^{4}SCSI disk driver that implements the disk request scheduling presented in Chapter 3, and the DROPS File System that provides the file-system properties described in Chapter 4. Both the disk driver and the file system are separate components of DROPS, using the programming environment provided by L^{4}ENV [28]. The L4 Environment consists of services and libraries that support the application development on top of the L4 microkernel, it particularly includes the management of low-level resources such as memory, threads and synchronization primitives.

On of the main components of the overall DROPS architecture is L^{4}LINUX [34], a paravirtualized version of the LINUX kernel running on top of the L4 kernel. L^{4}LINUX provides the main environment for timesharing applications in DROPS, allowing the execution of legacy applications within a fully-featured LINUX environment. Figure 5.1 illustrates two ways how L^{4}LINUX can be enabled to access the DROPS Disk-Storage System, both using stubs within L^{4}LINUX to connect either to the L^{4}SCSI disk driver or the DROPS File System. The current implementation uses the former approach, providing L^{4}LINUX with a new block device type to access the disks handled by L^{4}SCSI.

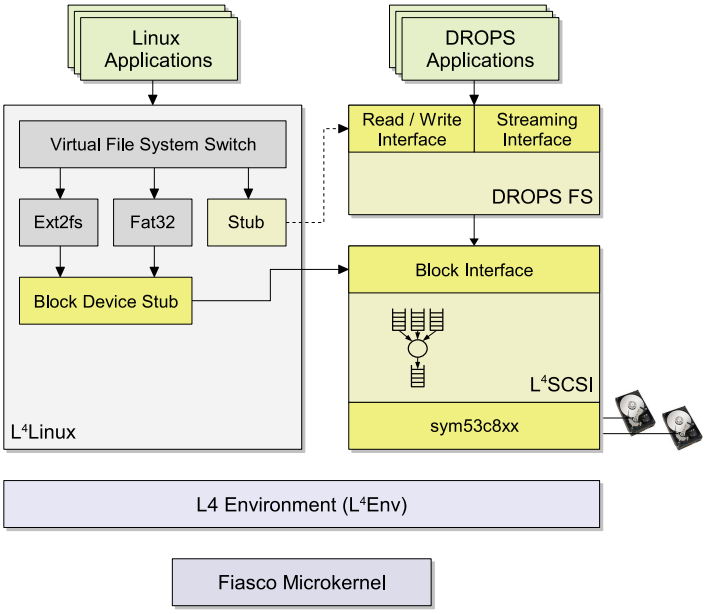


Figure 5.1: Overview of the DROPS Disk-Storage System.

5.1.1 L4SCSI

The L4SCSI disk driver implements the disk scheduling model described in Chapter 3, including the admission control to calculate the reservation times of real-time data streams. L4SCSI reuses LINUX device drivers to access the SCSI host adapter, using an emulation environment (DDE [35]) to provide the device driver with the required execution environment.

The client interface offered by L4SCSI provides an asynchronous execution model for disk requests, meaning that the call to submit requests to the driver immediately returns once the requests are enqueued within the driver and a separate call is used to notify the clients of the completion of the requests. Additionally, the client interface of L4SCSI contains functions to manage data streams, in particular to create and destroy streams and to start and stop the execution of a stream.

5.1.2 The DROPS File System

The DROPS File System deploys the data stream model provided by L4SCSI to read or write files with guaranteed bandwidths. The structure of the file system is based on the EXT2 file system [2], incorporating a modified allocation policy to enable the file allocation using various block sizes. The client interface of the DROPS File System provides both the streaming interface used by data streams and a traditional `read()` and `write()` interface.

5.2 Disk-Request Scheduling in L4SCSI

L4SCSI implements a periodic execution model for the requests of data streams. These requests are queued within the driver and are kept back until the beginning of the period in that these requests are executed according to the stream model. At the end of each period, L4SCSI removes all stream

requests not yet executed from the requests lists and provides the clients with an error code indicating that these requests were skipped. Best-effort requests and sporadic real-time requests are queued separately.

The actual disk-request scheduling in L^{4}SCSI consists of two steps that have to be executed each time a request needs to be selected to be issued to the disk. First, L^{4}SCSI must create the DAS, the subset of the outstanding requests that can be effectively included with the current scheduling decision, and second, a request scheduler must choose the request from this subset that is going to be executed.

5.2.1 Creating the DAS

L^{4}SCSI queues the requests of the data streams in separate queues, further dividing the requests of a stream into requests lists containing the requests of each individual period of the stream. This enables the efficient creation of the DAS by collecting the request lists of the streams that are allowed to be added to the DAS. This approach particularly does not require to inspect the individual requests of the streams. The listing shown in Figure 5.2 outlines the creation of the DAS implemented by L^{4}SCSI.

To create the DAS, L^{4}SCSI iterates over the list of streams, which is sorted according to the priorities of the streams starting with the stream with the highest priority. For a stream to be allowed to be added to the DAS, two conditions must be fulfilled:

1. Sufficient time must be available to not interfere with the execution of streams with a higher priority. To fulfill this condition, the remaining time until the end of the period is reduced by the amount of time required to execute each stream and the inclusion of further streams stops as soon as the remaining time does not allow the execution of a request with a lower priority (Line 36). Reserving the worst-case execution time guarantees that after the potential execution of a request with a lower priority still enough time is left to ensure the execution of the streams with a higher priority. The time required to execute a stream is defined by the utilization caused by the stream as discussed in Section 3.3.1 and depends on the type of the stream. With mandatory streams, the time is defined by the worst-case execution of each individual request (Line 26), with optional streams the time depends on the remaining budget of that stream (Line 32).

   If the set of data streams handled by L^{4}SCSI contains streams with different period lengths, the available time until the end of the period must be adjusted each time the creation of the DAS reaches a stream with a longer period (Line 13; streams with a longer period have a lower priority with the priority assignment discussed in Section 3.2.2). `update_remaining_time()` increases the available time until the end of the new period based on the number of shorter periods occurring until the end of the longer period and the available slack time provided by these shorter periods, as described in Section 3.3.

2. The stream must not have exceeded its budget (Line 28). For optional data streams, the budget available to a stream within a period is set to the reservation time a the beginning of each period, and the time spent to execute the requests of the stream is withdrawn from this budget. Mandatory data streams are always added to the DAS (Line 22).

Once L^{4}SCSI traversed all streams and there is still enough time left until the end of the period, additional requests such as sporadic real-time requests and best-effort requests can be added to the DAS (Line 43). At this place, a variety of policies are possible to decide how the remaining time is used:

- An acceptance test for sporadic real-time requests can utilize the remaining time to decide whether the system is able to accept these requests. As a result, the execution of the sporadic

```
1     /* streams       list of streams, sorted according to their priorities
2      * time_left     time left until the end of the shortest period
3      * wcet          worst-case execution time of a disk request
4      * das_streams   array of request-list pointers, it is filled with
5      *               the request lists of the streams added to the DAS */
6
7     s = streams;
8     period_length = s->period_length;
9     i = 0;
10    while (s)
11      {
12        /* update time_left if reached a stream with a longer period length */
13        if (s->period_length > period_length)
14          {
15            update_remaining_time(&time_left);
16            period_length = s->period_length;
17          }
18
19        /* get request list of the stream for the current period */
20        r = stream_requests(s);
21
22        if (s->quality == 1)
23          {
24            /* always add requests of a mandatory stream */
25            das_streams[i++] = r;
26            time_left -= (number_of_requests(r) * wcet);
27          }
28        else if (s->budget > 0)
29          {
30           /* only add requests of an opt. stream if it did not yet exceeded its budget */
31           das_streams[i++] = r;
32           time_left -= (s->budget + wcet);
33          }
34
35        /* stop adding requests if no more time is left in the period */
36        if (time_left < wcet) break;
37
38        /* continue with next stream */
39        s = s->next;
40      }
41
42    /* if enough time is left, add sporadic real-time or best-effort requests */
43    if (time_left > wcet)
44      ...
```

Figure 5.2: Creating the Dynamic Active Subset (DAS). The listing outlines the principle procedure used by L^{4}SCSI to create the DAS. Streams are added to the DAS as long as there is sufficient time left in the period and the streams did not yet fully consumed their budget. With the DAS being constructed by adding the entire requests lists of the streams (Lines 25 and 31), the complexity of the creation of the DAS depends only on the number of streams handled by L^{4}SCSI, and particularly not on the number of requests contained in the request lists.

real-time requests can be intermingled with the execution of the stream requests or the execution of stream requests can be even suspended to meet the deadlines of the sporadic real-time requests. Of course this can only happen to the extent allowed by the remaining slack time of the streams.

- Similar to the acceptance of sporadic real-time requests, the remaining time can be used to incorporate the execution of best-effort requests. Best-effort requests can be even prioritized over stream requests to achieve short response times, similar to the ΔL scheduler [11].

- Finally, requests of data streams initially not added to the DAS due to an exceeded budget can be added to further increase their quality.

The current implementation adds best-effort requests, but without prioritizing these requests and allows to add requests of data streams that already exceeded their time budget for the current period.

5.2.2 Selecting a Disk Request

With the guarantees of real-time streams and sporadic real-time requests being enforced by the DAS, the scheduling policy used to pick a request out of the DAS is released from any constraints regarding the enforcement of the guarantees. L4SCSI implements several scheduling policies, including the CSCAN [74], *Shortest Seek Time First* (SSTF) [74] and the *Shortest Access Time First* (SATF) [67, 63] policies.

The SATF scheduling policy picks the request out of the DAS that results in the shortest overall access time. The access time includes the time required to move the disk head to the target track (*seek time*), the time required to wait until the target sector arrives at the disk head (*rotational delay*) and the time caused by the data transfer and the command processing. To calculate the access time, the rotational position of both the current position of the disk head and the target sector need to be known. With the position of the disk head derived from the target sector of the preceding request, the calculation of the rotational position requires the mapping of the logical addresses used by the SCSI interface to the actual position on the disk, defined by the cylinder number and angle of the sector. This mapping and the functions describing the seek time depending on the track distance and the rotational delay depending on the angle distance are obtained using microbenchmarks and are used by L4SCSI to calculate the access times [63].

The CSCAN and SSTF scheduling policies do not require the knowledge of the rotational position, both choose the request based on the logical address of the sector, assuming a linear mapping of the logical address to the cylinders at the disk.

5.2.3 L4LINUX Block-Device Stub

The block-device stub is required to allow L4LINUX application to access the disks handled by L4SCSI. The stub implements a new block device in L4LINUX (`/dev/l4bxx`), which provides L4LINUX with the usual ways to access the disks handled by L4SCSI, in particular the ability to mount partitions on these disks. To execute the disk requests issued by L4LINUX for theses block devices, the stub removes the requests from the L4LINUX request queues and forwards them to L4SCSI, and a pseudo interrupt thread within L4LINUX is used to receive the completion notifications from L4SCSI and to wakeup the L4LINUX processes waiting for those requests. To provide L4SCSI with the target or source buffers of the requests, the current implementation includes the physical addresses of the buffers in the requests sent to L4SCSI , allowing the disk driver to transfer the data using *Direct Memory Access* (DMA). To eliminate the security risks associated with direct use of the physical addresses, the stub could incorporate a more sophisticated approach to provide L4SCSI with the target or source buffers [38], which is based on the memory management provided by L4ENV.

The current version of the block-device stub bases on the L^{4}LINUX version 2.2.26. It can be applied to more recent versions of L^{4}LINUX adopting the request handling within the stub to the changes in the block layer of the more recent LINUX versions.

5.3 The DROPS File System

The disk layout of the DROPS File System bases on the EXT2 file system [2] and is implemented using the EXT2FS library provided with the E2FSPROGS [1] package to manage the file system metadata. To provide the file-system properties described in Chapter 4, the DROPS File System incorporates a modified allocation policy with the EXT2 file system and implements the streaming interface and execution model described in Section 4.3.

Using an established file system as the basis enables the DROPS FS to provide a fully-featured file system, including directories, links and so on.

5.3.1 Block Allocation Supporting Various Block Sizes

The EXT2 file system divides the available disk space into *block groups*, closely following the ideas of the UNIX FFS [53] to store the file data close to its file-system metadata (i.e., the inode describing the file). The DROPS FS takes advantage of this structure by mapping the allocation groups used to provide the different block sizes to the block groups of the EXT2 file system. The allocation policy implemented with the DROPS FS ensures that each block group is only used to provide a single block size, tagging the block group in its superblock with the block size at the time the block groups is used for the first time.

The structure of the file-system metadata is not changed, in particular the structure is still based on a fixed block size, which typically is 4 KByte for a EXT2 file system. With the block groups providing a larger block size, this results in both an oversized block allocation bitmap and inode block lists. On the other hand, this provides a backward-compatibility with the EXT2 file system, particularly allowing the use of the file system initialization and maintenance tools.

5.3.2 The DROPS Streaming Interface

The design of the streaming interface discussed in Section 4.3 bases on a circular buffer created between the sender and the receiver of the stream. This approach aims

- to provide an appropriate synchronization mechanism, including the signalling of gaps within the stream caused by requests not executed for optional data streams;
- to ensure the correct order of the elements of the buffer; and
- to allow a zero-copy data transfer by sharing the buffer memory between the sender and the receiver.

This approach is implemented by the DROPS *Streaming Interface* (DSI) [51], which provides the main mechanism for the bulk transfer of data between DROPS components. Figure 5.3 outlines the design of the DSI and the usage of the interface by the DROPS FS to deliver data streams.

The circular buffer is implemented using two separate memory areas, a data area storing the actual data of the stream and a metadata area containing the descriptors used to construct the circular

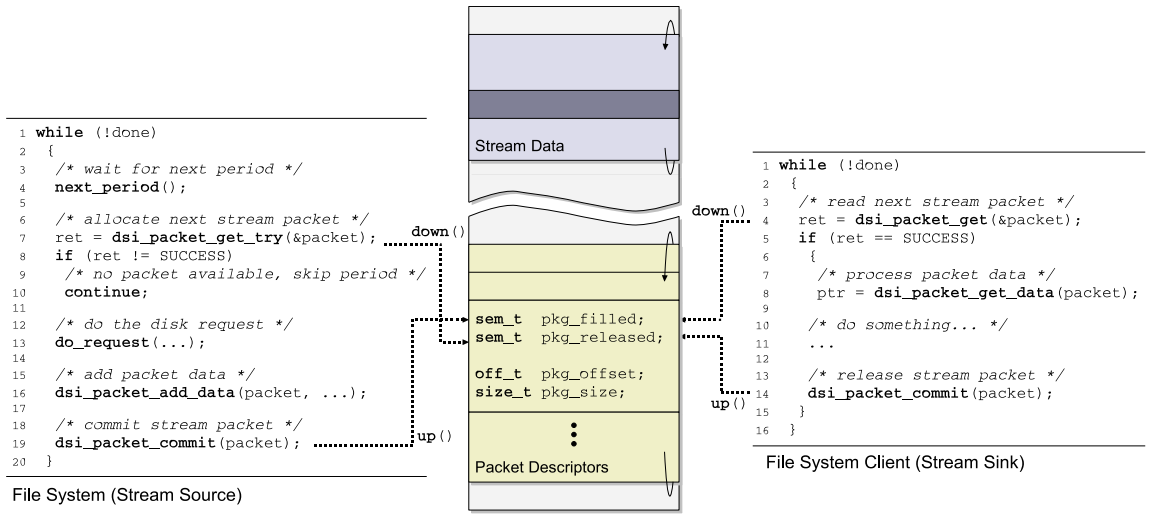


Figure 5.3: DROPS FS and the DROPS Streaming Interface (DSI). The figure illustrates both the design of the DSI and the usage of the DSI with the file system. The DSI uses two memory areas, both shared between the source and the sink of the stream. The data area contains the actual stream data, the descriptor area contains the descriptors used to describe the packets that are used to construct the stream. A packet descriptor consists of a reference to the data area and semaphores to synchronize both the entry of the data by the stream source and the release of a packet by the stream sink. To deliver a stream to a client, DROPS FS periodically adds packets to the stream containing the file data. The actual procedure is a bit more complex, as several packets are added to the stream within a period and the disk requests are executed asynchronously, meaning that all requests are submitted to L^{4}SCSI at the beginning of a period and the results are checked at the end of the period.

buffer. Both the sender and the receiver of the data stream can access these memory areas using the memory management mechanisms offered by L^{4}ENV, providing the zero-copy transfer of the data between the sender and the receiver of the stream. Storing the metadata of the buffer outside the data area allows the client to contiguously access the buffer elements, which can be used to provide the notion of a sliding window over the buffer[1].

The circular buffer is constructed of several packets, each packet describing a single element of the buffer. Both the sender and the receiver of a stream sequentially process the packets, establishing the order of the buffer elements. The data structure describing a packet consists of two main parts, a reference to the data area and synchronization primitives.

The reference to the data area is stored relative to the beginning of that area, which allows to specify the data independently of the position of the data area within the address spaces of the sender and the receiver. To signal a gap within the stream, a packet is tagged with a flag indicating the gap. Similarly, the receiver of the stream is notified about the end of the stream by tagging the final packet of the stream with a flag indicating the end.

The access to the packets must be synchronized to ensure that the receiver only accesses valid data as well as that the sender does not overwrite data not yet consumed by the receiver. Both of these requirements are achieved using semaphores implemented within each packet descriptor.

The code fragments shown in Figure 5.3 also provide a simplified overview of the use of the DSI by the DROPS file system to deliver a data stream to a client. The file system periodically allocates the

[1] A sliding window requires a proper handling of the wrap-around at the end of the buffer, which can be achieved by remapping the beginning of the buffer behind the end of the buffer.

next element of the buffer and executes the disk request to read the next section of the file. The client consumes the data and eventually releases the packet, which makes the packet again available for the file system. The file system skips the creation of the disk request if no buffer element is available (Line 8). This situation particularly happens if the file system reads the data at a higher rate than the client consumes the data, which can be an intended behavior to compensate the variations of data streams with variable bit rate demands as described in Section 3.1.1. By skipping the creation of further disk requests, the file system avoids a buffer overflow in these situations. The approach works likewise for data streams written by the file system. The file system periodically takes elements out of the buffer and creates the according disk requests. If the buffer is empty, the file system obviously creates no disk requests.

The actual implementation with the DROPS FS is a bit more complex compared to the fragment shown in Figure 5.3, because the file system executes several requests within a period. The period length used by the file system is equal to the period length of the data stream handled by L^4SCSI to read or write the data stream. The file system submits all requests required to process the stream to L^4SCSI using the asynchronous interface of L^4SCSI at the beginning of each period, and checks the results of the execution at the end of the period. For data streams delivered to the client, the file system either adds the data to the DSI packet if the disk request was successfully executed or tags the packet indicating a gap in the stream if the request was skipped for an optional stream. The file system metadata required to create the disk requests is completely held in memory by the file system, requiring no additional disk requests during the processing of the stream.

5.4 Summary

This chapter provided an overview of the implementation of the DROPS Disk-Storage System, which incorporates both the disk-request scheduling and file system supporting quality-of-service guarantees. In particular, the chapter documented the practicability of the disk request scheduling (Section 5.2) and the file system design (Section 5.3).

The following chapter provides an experimental evaluation of the presented designs using the implementations described in this chapter.

Chapter 6

Experimental Evaluation

The previous chapters presented the designs of a disk-storage system and a file system that both support quality-of-service guarantees and the application of these designs to the DROPS Disk-Storage System. The following chapter contains an experimental evaluation of those designs. The evaluation aims to examine the accuracy of the admission model, to point out the benefits of the presented designs, and to analyze the influence of the enforcement of the quality-of-service guarantees on the overall system performance. In particular, the evaluation

- examines the accuracy with that the disk scheduling meets the predictions of the admission model and analyzes the influence of the characteristics of the request-execution-time distributions on the accuracy;
- points out the detailed benefits of the reservation-based disk scheduling, the use of statistical guarantees, the use of the DAS scheduler, and the block allocation supporting several block sizes;
- analyzes the effect of the enforcement of the guarantees on the disk bandwidth; and
- analyzes the costs of the integration of the storage system into the DROPS architecture.

Section 6.1 describes the evaluation environment, in particular the characteristics of the disk drives and workloads used throughout the evaluation. Section 6.2 examines the accuracy of the stream qualities achieved by disk scheduling and analyzes the benefits and costs of enforcing the quality-of-service guarantees. Section 6.3 discusses the effects of the Dynamic Active Subset (DAS) and Section 6.4 examines the costs of integrating the storage system into the overall DROPS architecture. Section 6.5 evaluates the design of the file system. Finally, Section 6.6 concludes the evaluation with a summary.

6.1 Evaluation Environment

All experiments were performed using SCSI disks connected to a PC-based system consisting of a 1 GHz Intel Pentium 3 processor, Intel 815EP chipset, 256 MB main memory and a Tekram DC-390U3W SCSI host adapter [86]. The SCSI host adapter is based on the LSI Logic / Symbios Logic sym53c1010 Ultra3 SCSI chipset, which allows a maximum transfer rate of 160 MByte/s between the disk and the host adapter.

6.1.1 Disk Drive Parameters

IBM Ultrastar 36Z15 IC35L018UWPR15 [40]	
Size	18.4 GByte
Revolutions per Minute (RPM)	15 000
$t_{maxseek}$	7.178 ms
n	5
t_{rot}	4.000 ms
t_{sector}	0.011 ms
m (64 KByte Request Size)	128
t_{skew}	0.994 ms
v	1
t_{ovh}	0.671 ms
Worst-Case Execution Time w	30.251 ms

Seagate Cheetah 36ES ST318406LW [73]	
Size	18.4 GByte
Revolutions per Minute (RPM)	10 000
$t_{maxseek}$	10.938 ms
n	4
t_{rot}	5.971 ms
t_{sector}	0.011 ms
m (64 KByte Request Size)	128
t_{skew}	4.095 ms
v	1
t_{ovh}	0.436 ms
Worst-Case Execution Time w	40.761 ms

Table 6.1: Disk Drive Parameters. The worst-case execution times are calculated based in the model 3.36 presented in Section 3.2.3 and the required disk parameters are obtained using microbenchmarks [62].

Two disks were used to perform the experiments, an IBM Ultrastar 36Z15 disk drive and a Seagate Cheetah 36ES disk drive. Table 6.1 shows the parameters of the disk drives and the resulting worst-case execution times based on the model presented in Section 3.2.3. With an average-case execution time in the range of 0.5 ms–6 ms for the measurements presented throughout the evaluation, the worst-case execution exceeds the average-case execution time by about an order of magnitude.

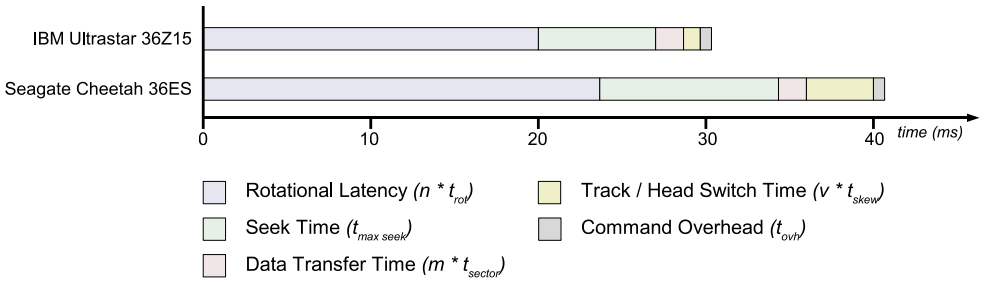
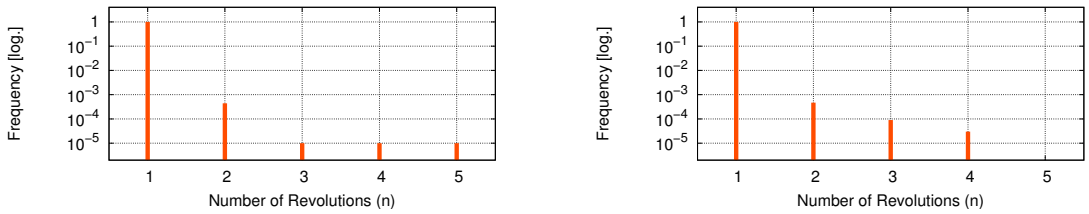


Figure 6.1: Composition of the Worst-Case Execution Time. About two-thirds of the worst-case execution time of a disk request is caused by the rotational latency.

Figure 6.1 depicts the distribution of the worst-case execution time to the single elements of the worst-case execution model. The results show that about two-thirds of the worst-case execution time is caused by the rotational latency required to settle the disk head to the target sector. The large rotational latency results from additional revolutions required by the disk if the disk head misses the target track initially. Figure 6.2 shows the number and frequency of the revolutions measured in a worst-case scenario [62].

A common approach to improve the worst-case behavior of a disk drive is to consider a set of requests instead of a single disk request to reduce the overall seek time. For instance, the Fellini Multimedia Storage System [52] schedules the requests of a whole period according to the CSCAN policy and considers only one full seek of the disk head to calculate the execution time of the whole request set. However, the effectiveness of these approaches is limited, as the worst-case execution time is to a substantial part caused by the rotational latency that occurs for each disk request separately.

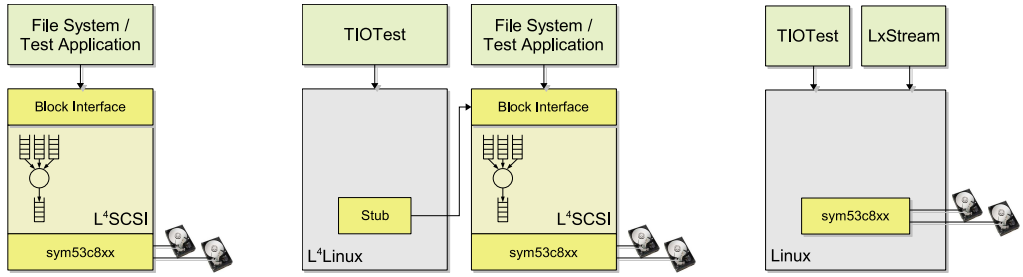


(a) IBM Ultrastar 36Z15 (b) Seagate Cheetah 36ES

Figure 6.2: Number and frequency of the disk revolutions measured for the rotational latency. If the disk requires more than one revolution (i.e., $n > 1$), the disk head missed the target track initially.

6.1.2 Evaluation Setups

Figure 6.3 illustrates the various setups used by the experiments throughout the evaluation.



(a) L^{4}SCSI (b) L^{4}LINUX + L^{4}SCSI (c) LINUX

Figure 6.3: Evaluation setups used throughout the evaluation.

The main setup used in the evaluation is shown in Figure 6.3(a). A test application directly issues requests to L^{4}SCSI in the same way the file system does. The requests are generated by the test application using block lists describing the workload of the experiment. L^{4}SCSI uses the SATF policy to schedule the requests of the DAS.

Figure 6.3(b) shows the setup used to analyze the integration of the storage system into the DROPS architecture. Besides L^{4}SCSI, a disk benchmark (TIOTEST [3]) is executed on L^{4}LINUX. L^{4}LINUX uses a stub driver connected to L^{4}SCSI to access the disk drives.

Finally, Figure 6.3(c) shows the setup used to obtain performance values that can be achieved using off-the-shelf systems. Instead of a client application using L^{4}SCSI, data streams are read by a normal LINUX application (LXSTREAM) using the standard LINUX disk driver.

6.1.3 Benchmarking Workloads

The results of a file system or disk benchmark are significantly influenced by the workload used in the benchmark experiments. The workload defines the characteristics of the disk requests handled by the storage system, for instance whether requests belong to a contiguous area of the disk or if the requests are spread over various areas of the disk. With these effects, the workload significantly influences the execution times of the disk requests.

Two properties are required to describe a workload, the access pattern to a file and the allocation pattern of the file on the disk. It is important to consider both properties, as a sequential file access might still cause the disk requests to be distributed over the whole disk if the disk blocks of the file are not allocated contiguously.

For the experiments presented in this chapter, the file access pattern is determined by the data stream model. The files are accessed contiguously, and the accesses are performed periodically.

The allocation pattern of a file results from the allocation policy of the file system and is mainly influenced by the fragmentation of the file system space. Most file systems attempt to allocate files contiguously, but this only succeeds to the extent to that contiguous free space is available on the disk. Thus, the disk blocks of a file created on an empty file system are allocated contiguously, whereas the disk blocks might be spread over the available disk space on an aged file system. To acquire reasonable benchmark results, the workloads used in the benchmarks should reflect these properties [80, 81].

The workloads used throughout this evaluation were obtained recording the allocation patterns of files created on two different EXT2 file systems. One was the user file system of a local workstation, which was one year old and about 50 % utilized, the other was the file system of a workgroup server, which was 3 years old and nearly full utilized (99 %). Instead of using the absolute block numbers, the patterns were recorded by tuples describing the number of contiguous disk blocks in a file fragment and the distance to the next file fragment. This allows to apply the patterns to files created on file systems with a different size (aligning the resulting block numbers to the size of the file system) and to files using a different allocation block size. Table 6.2 shows the characteristics of the recorded file patterns, together with the characteristics of a linear allocation and a random allocation that evenly distributes the blocks of the file over the available disk space.

With the linear file allocation (Table 6.2(a)), the file fragments are disrupted only by the disk blocks storing the block list of the file and the blocks storing the header of a block group. The pattern recorded on the workstation file system (Table 6.2(c)) consists of parts that are allocated similar to the linear pattern as well as parts consisting of less contiguous blocks, resulting from the fragmentation of the file system. The pattern recorded on the workgroup server (Table 6.2(d)) contains almost no large contiguous fragments, instead the blocks are scattered over the whole file system as a result of the high fragmentation of the nearly full file system.

To create the actual workloads used in the evaluation, four files with a size of 512 MB were created on both the workstation file system and the workgroup-server file system. Table 6.3 shows the characteristics of the recorded patterns. The workstation file system represents a file system with a normal level of fragmentation, the patterns are named *Pattern-nf*. The workgroup-server represents a highly fragmented file system, the patterns are named *Pattern-hf*.

For each evaluation scenario, a set of files were created on a newly initialized EXT2 file system using the recorded patterns. The allocation groups used to support different block sizes are mapped to the EXT2 block groups, each EXT2 block group contains blocks of a fixed block size.

The L^4SCSI client application uses block lists obtained from those files to create the disk requests, TIOTEST and LXSTREAM directly access the files on the file system.

File Offset	Contiguous Blocks	Distance to Next Block
0	12	2
12	1024	1
1036	1024	1
2060	1024	1
	...	
31756	976	6
32732	48	1
32780	1024	1
	...	

(a) Linear File Pattern.

File Offset	Contiguous Blocks	Distance to Next Block
0	1	68730
1	1	2692134
2	1	2556781
3	1	1303004
4	1	161305
5	1	1966063
6	1	161445
7	1	1256637
	...	

(b) Random File Pattern.

File Offset	Contiguous Blocks	Distance to Next Block
	...	
46	6	2145
52	52	1
	...	
18018	1	49
18019	6	336
	...	
20073	419	1
20492	1024	1
21516	37	1
	...	

(c) Workstation File Pattern.

File Offset	Contiguous Blocks	Distance to Next Block
	...	
2491	22	6
2513	23	2
	...	
18165	2	14931
18167	12	2
18179	14	1
	...	
32121	5	136031
32126	113	72
	...	

(d) Server File Pattern.

Table 6.2: File Allocation Patterns. The tables show the characteristics of the patterns, that is the number of contiguous blocks in a file fragment and the distance to the next file fragment (in number of blocks).

Pattern	Number of Fragments	Avg. Fragment Size	Avg. Distance
Linear 1	133	985.5	1.1
Linear 2	133	985.5	1.1
Linear 3	133	985.5	1.1
Linear 4	133	985.5	1.1
Random 1	131072	1.0	1496825.4
Random 2	131072	1.0	1491688.7
Random 3	131072	1.0	1492884.5
Random 4	131072	1.0	1493111.3

Pattern	Number of Fragments	Avg. Fragment Size	Avg. Distance
Pattern-nf 1	3385	38.7	248.2
Pattern-nf 2	1680	78.0	1966.3
Pattern-nf 3	1905	68.8	1218.8
Pattern-nf 4	1822	71.9	1880.1
Pattern-hf 1	4566	28.7	1659.6
Pattern-hf 2	8730	15.0	1466.7
Pattern-hf 3	8380	15.6	1618.1
Pattern-hf 4	9819	13.3	2622.0

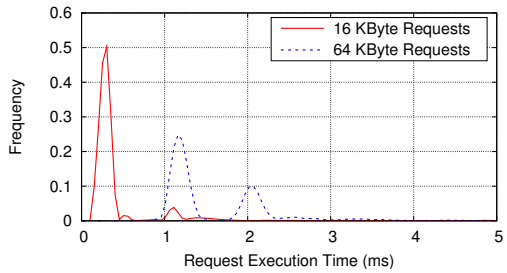
Table 6.3: Benchmarking Workloads. The tables show the number of file fragments, the average size of a fragment (in blocks) and the average distance to the next file fragment (in blocks) of the workloads used in the evaluation.

6.2 Disk Scheduling with Quality-of-Service Guarantees

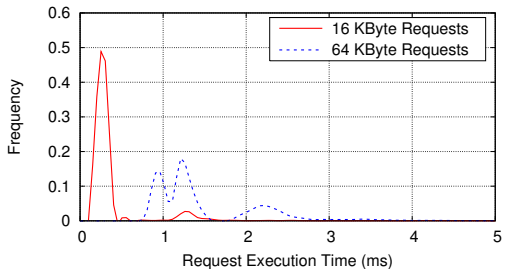
This section examines the feasibility of the stream admission model. The evaluation contains an analysis of the accuracy of the achieved stream qualities, an illustration of the benefits of the reservation-based disk scheduling using statistical guarantees, and an analysis of the influence of the enforcement of the guarantees on the available disk bandwidth.

6.2.1 Accuracy of the Achieved Stream Qualities

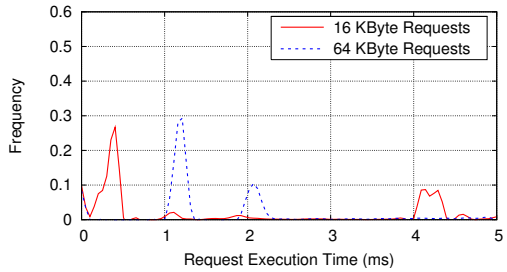
To examine the accuracy with that the disk scheduling meets the predicted stream qualities, the test application executes a setup consisting of several data streams and an additional best-effort load that consumes the disk bandwidth not used by the data streams. Figure 6.4 contains the request-execution-time distributions used to calculate the reservation times of the data streams, the distributions are obtained executing the test setup in a calibration run. The distributions are obtained for each workload separately, Figure 6.4 shows the distributions for the Pattern-nf workload.



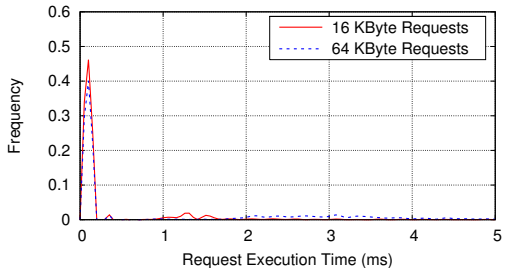
(a) IBM Ultrastar 36Z15, Read Requests.



(b) Seagate Cheetah 36ES, Read Requests.



(c) IBM Ultrastar 36Z15, Write Requests.



(d) Seagate Cheetah 36ES, Write Requests.

Request Size	IBM Ultrastar 36Z15						Seagate Cheetah 36ES					
	Read			Write			Read			Write		
	mean	dev	max	mean	dev	max	mean	dev	max	mean	dev	max
16 KByte	0.50	0.73	20.10	1.75	1.96	12.40	0.53	1.00	13.30	3.23	3.38	15.80
64 KByte	1.64	0.78	11.20	2.06	1.76	24.00	1.60	1.04	37.00	3.78	3.72	32.10

(e) Distribution Parameters Mean Value, Standard Deviation, and Maximum Value (in ms).

Figure 6.4: Request-execution-time distributions used to calculate the stream reservation times for the Pattern-nf workload.

Table 6.4 presents the results for a test setup reading streams from the disks and Table 6.5 presents the results for a test setup writing streams to the disks. The setups were chosen such that they form the maximum possible setup accepted by the admission control, meaning that an increase of the

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
1	1000	24	64	1.00	726.02	1.00	0.0	1536.0
2	1000	184	64	0.95	286.00	0.93	-2.1	10912.9
3	1000	208	64	0.90	306.20	0.91	1.1	12091.3
4	1000	224	64	0.77	282.00	0.77	0.0	11052.8
5	4000	34	16	1.00	1028.53	1.00	0.0	136.0
6	4000	184	16	0.95	89.00	0.98	3.2	720.5
7	4000	208	16	0.90	92.70	0.95	5.6	787.8
8	4000	224	16	0.72	94.50	0.78	8.3	697.9
						Average Difference Δq_{dev}	2.5	
						Mean Value $\overline{q_{dev}}$	2.0	
						Standard Deviation σ	3.2	
						Best-Effort Requests (64 KByte Request Size)		61.6
						Best-Effort Requests (16 KByte Request Size)		38.5
						Total Bandwidth		38035.3
						Achieved Disk Utilization of Real-Time Streams		1.00

(a) IBM Ultrastar 36Z15.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
1	1000	16	64	1.00	652.18	1.00	0.0	1024.0
2	1000	192	64	0.95	292.80	0.92	-3.2	11321.0
3	1000	208	64	0.90	299.30	0.90	0.0	12033.7
4	1000	248	64	0.72	285.40	0.73	1.4	11579.5
5	4000	16	16	1.00	652.18	1.00	0.0	64.0
6	4000	192	16	0.95	99.90	0.94	-1.1	720.5
7	4000	208	16	0.90	98.50	0.92	2.2	763.2
8	4000	248	16	0.75	112.80	0.83	10.7	824.1
						Average Difference Δq_{dev}	2.3	
						Mean Value $\overline{q_{dev}}$	1.3	
						Standard Deviation σ	3.9	
						Best-Effort Requests (64 KByte Request Size)		150.0
						Best-Effort Requests (16 KByte Request Size)		195.6
						Total Bandwidth		38675.6
						Achieved Disk Utilization of Real-Time Streams		0.99

(b) Seagate Cheetah 36ES.

Table 6.4: Achieved Stream Qualities and Disk Utilization, Stream Read, Pattern-nf Workload.

quality of a stream of the setup would result in the rejection of the setup by the admission control. This way of creating the benchmark setups results in different bandwidths and stream qualities used with the two disks and the different workloads, as the request-execution-time distributions used to calculate the reservation times are influenced by both the disk characteristics and the workloads. Requests of optional data streams that exceeded the reservation time for a period were dropped.

The results show that the mandatory data streams (i.e., the streams with a requested quality of 1) are completely executed and that, with the exception of the last stream, the optional streams achieve their quality with a maximum relative deviation of about 5 %. The relative deviation is defined by

$$q_{dev,i} = \frac{q_{ach,i} - q_i}{q_i} \tag{6.1}$$

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
1	1000	24	64	1.00	726.02	1.00	0.0	1536.0
2	1000	112	64	0.95	221.00	0.93	-2.1	6631.5
3	1000	136	64	0.90	248.40	0.93	3.3	8102.2
4	1000	168	64	0.85	289.40	0.88	3.5	9442.4
5	4000	34	16	1.00	1028.53	1.00	0.0	136.0
6	4000	112	16	0.95	190.80	0.94	-1.1	420.8
7	4000	136	16	0.90	214.00	0.93	3.3	503.9
8	4000	168	16	0.79	269.20	0.80	1.3	540.9
						Average Deviation Δq_{dev}	1.8	
						Mean Value $\overline{q_{dev}}$	1.0	
						Standard Deviation σ	2.0	
						Best-Effort Requests (64 KByte Request Size)		69.1
						Best-Effort Requests (16 KByte Request Size)		18.6
						Total Bandwidth		27401.3
						Achieved Disk Utilization of Real-Time Streams		1.00

(a) IBM Ultrastar 36Z15.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
1	1000	16	64	1.00	652.18	1.00	0.0	1024.0
2	1000	64	64	0.95	239.80	0.99	4.2	4038.2
3	1000	72	64	0.90	247.40	0.92	2.2	4219.1
4	1000	84	64	0.85	269.60	0.87	2.4	4701.0
5	4000	16	16	1.00	652.18	1.00	0.0	64.0
6	4000	64	16	0.95	204.40	0.99	4.2	252.8
7	4000	72	16	0.90	210.50	0.90	0.0	257.9
8	4000	84	16	0.77	259.30	0.90	16.9	302.1
						Average Deviation Δq_{dev}	3.7	
						Mean Value $\overline{q_{dev}}$	3.7	
						Standard Deviation σ	5.2	
						Best-Effort Requests (64 KByte Request Size)		8.1
						Best-Effort Requests (16 KByte Request Size)		1.3
						Total Bandwidth		14868.5
						Achieved Disk Utilization of Real-Time Streams		1.00

(b) Seagate Cheetah 36ES.

Table 6.5: Achieved Stream Qualities and Disk Utilization, Stream Write, Pattern-nf Workload.

where q_i is the requested quality of the data stream and $q_{ach,i}$ is the measured quality of the stream. The large deviation of the quality of the stream with the lowest priority is a result of the approximation described in Section 3.2.2 to consider requests that overlap with the new period at the end of a period. Reducing the available time for the admission by the worst-case execution time of a request causes the admission control to assume that the execution of a stream is preempted at the end of a period more often than it actually happens with the real scheduling. This leads to the calculation of a reservation time for this stream that is larger than required to achieve the requested quality, causing the stream to achieve a higher quality than requested.

The results presented in Table 6.4 and Table 6.5 also show that the available disk bandwidth can be almost fully utilized by the data streams, denoted by the low bandwidth achieved by the best-effort load that consumes the time nor used by the data streams.

The setups used in the first experiment consists of data streams with arbitrarily chosen bandwidths, which have no relation to any real-world application scenario. Appendix A.1 contains the results of the same experiment using a more complex setup with reasonable stream bandwidths. To be able to compare the overall results of the experiments, the results are characterized by the average deviation Δq_{dev} and the mean value $\overline{q_{dev}}$ and standard deviation σ of the quality deviation.

The average deviation Δq_{dev} is defined by

$$\Delta q_{dev} = \frac{1}{n}\sum_{i=1}^{n} \left| q_{dev,i} \right| \tag{6.2}$$

and indicates the average deviation that occurs for the individual streams.

The mean value $\overline{q_{dev}}$, which is defined by

$$\overline{q_{dev}} = \frac{1}{n}\sum_{i=1}^{n} q_{dev,i}, \tag{6.3}$$

provides an indication of the quality of the overall scheduling.

The standard deviation σ is defined by

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (\overline{q_{dev}} - q_{dev,i})^2} \tag{6.4}$$

Table 6.6 summarizes the results of the experiments presented in Appendix A.1. To obtain the true accuracies of the qualities achieved by the disk scheduling, the setups used in these experiments avoid the inaccuracy for low prioritized streams described earlier by not fully utilizing the disk bandwidth, the average utilization used by these setups is 95 %. The results show the largest average deviation (5 %) for the setup using the Pattern-hf workload, indicating an influence of the high fragmentation of that workload on the quality of the streams.

	IBM Ultrastar 36Z15			Seagate Cheetah 36ES		
Workload	Δq_{dev}	$\overline{q_{dev}}$	σ	Δq_{dev}	$\overline{q_{dev}}$	σ
Linear	3.9 %	1.5 %	4.9	3.6 %	1.7 %	4.4
Pattern-nf	2.6 %	0.9 %	3.6	3.4 %	0.2 %	4.8
Pattern-hf	5.0 %	1.3 %	6.6	5.3 %	1.7 %	6.4
Random	1.7 %	0.4 %	2.7	1.4 %	0.0 %	2.2

Table 6.6: Results of the experiments using a complex setup. Appendix A.1 contains the full results of the experiments.

Effect of the Workload Used in the Admission Control

The experiments presented until now used setups based on an admission control using distributions that exactly represented the workload used in these experiments. All distributions were obtained performing a calibration run of the setup using the identical workload of the experiment. However, in real-world environments such accurate distributions might not be available. To assess the possible error caused by an admission based on inaccurate execution-time distributions, the experiments presented in Appendix A.1 were repeated. But instead of using the setups based on the actual distributions of the workload, the experiments used the setups based on the distributions of the random workload and Pattern-hf workload, as these distributions could be used as a pessimistic approximation of the request-execution-time distributions. With the more conducive workloads, this approximation results in the calculation of reservation times that are larger than required to achieve the

Stream Workload	IBM Ultrastar 36Z15	Seagate Cheetah 36ES
Linear	0.33	0.24
Pattern-nf	0.38	0.29
Pattern-hf	0.41	0.30

(a) Disk utilization achieved for the setups based on the random workload.

Stream Workload	IBM Ultrastar 36Z15	Seagate Cheetah 36ES
Linear	0.84	0.80
Pattern-nf	0.91	0.90

(b) Disk utilization achieved for the setups based on the Pattern-hf workload.

Table 6.7: Disk utilization depending on the admission workload. The tables show the achieved disk utilization reading streams with the various workloads. The setups of the experiments (i.e., the number of streams, bandwidths and qualities) are not based on the actual workload executed by the streams, instead, the setups base on the admission calculated for the random workload (Table (a)) and for the Pattern-hf workload (Table (b)).

requested qualities with these workloads. In fact, most of the streams using the linear and Pattern-nf workload achieved a quality of 1 (i.e. all requests of the streams were executed). Another effect of the approximation is that the streams do not fully utilize the available bandwidth, although the admission control is not able to admit more streams. Table 6.7 shows the utilization of the disk bandwidth achieved for the various stream workloads executing the setups based on the random and Pattern-hf workload. The results show that using the random workload to approximate the request-execution time is too pessimistic for the real-world workloads, even the streams using the highly fragmented workload do only achieve a utilization of 41 % and 30 %. Using the distributions of the Pattern-hf workload is a more reasonable approximation, it results in a better utilization achieved by the data streams.

Effect of the Distribution Class Width

Another property of the execution-time distributions that also effects the accuracy of the admission model is the class width of the distributions. The class width sets the precision of the calculated reservation time. Moreover, the class width largely affects the execution time of the admission control itself. The class width determines the number of elements contained in the distribution, which is a major component of the computational complexity.

To analyze the influence of the distribution class width, the experiments with the complex setups were repeated, performing the admission control using distributions with various class widths. Table 6.8 summarizes the results of these experiments, Appendix A.2 contains the full results. The results show no serious effect of the distribution class width on the accuracy of the achieved stream qualities. The mean value $\overline{q_{dev}}$ of the deviation shows a small increase with the largest class width (in particular for the linear workload), indicating that the streams achieve a slightly higher overall quality. This increase is caused by a slightly larger reservation time calculated by the admission control as a result of the more coarse-grained execution-time distribution.

Table 6.8 also contains the time t_{adm} required to perform the admission control, indicating the computational complexity of the admission control. The admission test of the mandatory data streams (Eqn. 3.13) exhibits a linear complexity depending on the number of disk requests contained in the streams. But in addition to the actual admission test, the admission control needs to calculate the distribution of the aggregated execution time of all requests of the mandatory data streams (Eqn. 3.14), which increases the complexity to $O(av^2)$. a is the total number of disk requests of the mandatory streams and v^2 denotes the complexity of a single convolution (Eqn. 3.26), which depends on the number v of elements contained in the distributions.

	Class Width 0.10 ms				Class Width 0.20 ms				Class Width 0.40 ms			
Workload	t_{adm}	Δq_{dev}	$\overline{q_{dev}}$	σ	t_{adm}	Δq_{dev}	$\overline{q_{dev}}$	σ	t_{adm}	Δq_{dev}	$\overline{q_{dev}}$	σ
Linear	77.26 s	3.9 %	1.5 %	4.9	11.21 s	3.5 %	1.4 %	4.5	1.75 s	4.0 %	2.6 %	4.4
Pattern-nf	99.45 s	2.6 %	0.9 %	3.6	12.01 s	2.8 %	1.4 %	3.8	2.07 s	3.1 %	2.4 %	3.5
Pattern-hf	93.85 s	5.0 %	1.3 %	6.6	11.70 s	4.4 %	0.8 %	5.9	1.90 s	4.7 %	0.0 %	6.7
Random	11.74 s	1.7 %	0.4 %	2.7	2.02 s	1.9 %	0.5 %	2.5	0.35 s	1.9 %	0.5 %	2.5

(a) IBM Ultrastar 36Z15.

	Class Width 0.10 ms				Class Width 0.20 ms				Class Width 0.40 ms			
Workload	t_{adm}	Δq_{dev}	$\overline{q_{dev}}$	σ	t_{adm}	Δq_{dev}	$\overline{q_{dev}}$	σ	t_{adm}	Δq_{dev}	$\overline{q_{dev}}$	σ
Linear	202.20 s	3.6 %	1.7 %	4.4	31.44 s	3.6 %	1.7 %	4.4	4.71 s	3.5 %	2.9 %	3.7
Pattern-nf	235.28 s	3.4 %	0.2 %	4.8	32.03 s	3.0 %	0.8 %	4.4	4.39 s	4.1 %	1.9 %	5.8
Pattern-hf	175.56 s	5.3 %	1.7 %	6.4	23.73 s	4.7 %	1.3 %	5.8	3.48 s	4.7 %	2.0 %	6.0
Random	21.09 s	1.4 %	0.0 %	2.2	3.25 s	1.5 %	0.3 %	2.3	0.54 s	1.6 %	0.3 %	2.4

(b) Seagate Cheetah 36ES.

Table 6.8: Results of the experiments using distributions with various class widths to calculate the reservation times. Appendix A.2 contains the setups and full results of the experiments.

The admission control for an optional data stream consists of three main parts, the calculation of the reservation time r_i (Eqn. 3.21), the calculation of the resulting execution time Z_i (Eqn. 3.30) of the data stream and the calculation of the n-times convolution X_i' with $i = 1, \ldots, a_i$ (Eqn. 3.27) required to calculate both r_i and Z_i.

- Equation 3.21 to calculate r_i is solved using a binary intersection to find the minimum reservation time that provides the requested quality. The achievable accuracy of the reservation time depends on the class width of the distributions. To test the admission criteria for a given reservation time, the admission control needs to calculate the random variable $A_i(r)$ (Eqn. 3.28), which exhibits a complexity of $O(a_i v^2)$ with a_i specifying the number of requests of the data stream and v the number of elements contained in the distribution. Combined with the binary intersection, the complexity of solving Equation 3.21 is $O(a_i v^2 log_2 v)$.

- The complexity of the calculation of the resulting execution time Z_i (Eqn. 3.30) is determined by the calculation of the random variables U_{ij} (Eqn. 3.31–3.34), which exhibit a complexity of $O(a_i v^3)$.

- The complexity of the n-times convolution X_i' is $O(a_i v^2)$.

Altogether, the overall complexity of the admission control for an optional data stream is dominated by the calculation of Z_i, resulting in a cubical dependency of the complexity from the number of elements contained in the execution-time distributions. The measured admission times t_{adm} shown in Table 6.8 confirm this result.

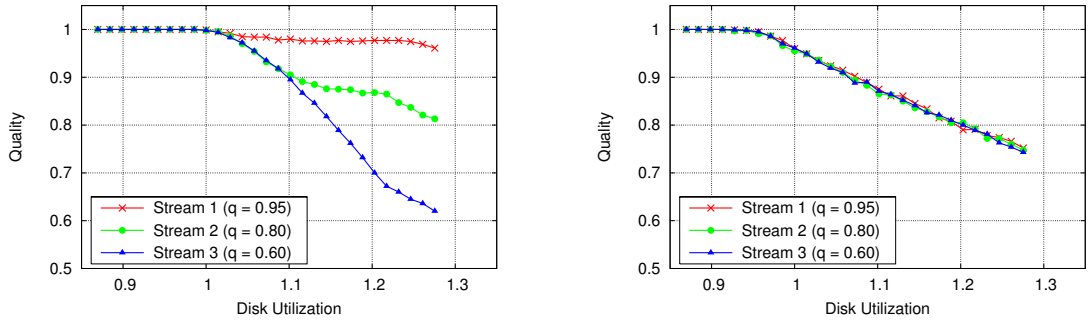
6.2.2 Benefits of the Quality-of-Service Guarantees

The results of the experiments presented until now document that the disk scheduling can meet the statistical guarantees with an adequate accuracy. The main motivation for such statistical guarantees is to both provide a deterministic behavior of the storage system in case of an overload situation and to improve the performance of the storage system compared to guarantees based on worst-case assumptions.

Overload Behavior

Overload situations can result from stream configurations exceeding the bandwidth that the disk is able to handle as well as greedy best-effort loads concurrently accessing the disk. In both situations, the disk scheduling must ensure that the guarantees given by the admission control are met. That means, all data streams achieve at least the requested qualities.

Figure 6.5 shows the behavior of data streams in case of the utilization caused by the data streams exceeds the abilities of the disk. As the utilization exceeds 1, the qualities of the streams start to decrease (the remaining requests of streams that exceeded their budget in a period were added to the DAS if possible, which results in a achieved stream quality of 1 if the disk is not fully utilized). With the enforcement of the guarantees by L^{4}SCSI (Fig. 6.5(a)), the loss of quality is distributed between the streams such that each stream still achieves its requested quality, whereas without such guarantees the loss is evenly distributed between the streams (Fig. 6.5(b)). The results show that the reservation-based enforcement of guarantees is able to provide an isolation between streams with respect to the stream qualities.



(a) Stream qualities with enforced guarantees (L^{4}SCSI).

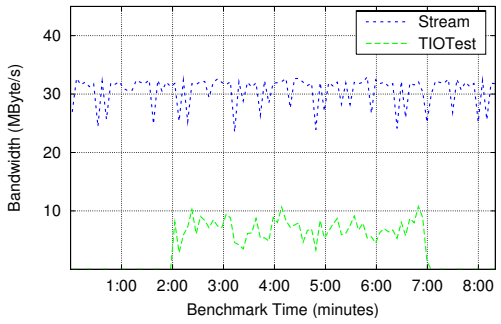
(b) Stream qualities without guarantees (LINUX).

Figure 6.5: Stream qualities in overload situations. Both experiments execute a setup of three streams reading files with the same bandwidth. Throughout the experiments, the bandwidth of the streams is increased to exceed the bandwidth that can be provided by the disk, creating an overload situation (i.e., the utilization demanded by the streams exceeds 1). Figure (a) shows the resulting stream qualities of the experiment using L^{4}SCSI to enforce the requested qualities. Figure (b) shows the results of the experiment running on LINUX, which provides no guarantees.

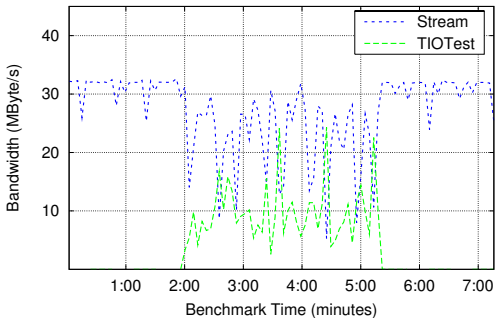
The reservation-based disk scheduling also provides the isolation of data streams and best-effort loads. Figure 6.6 shows the results of experiments concurrently executing data streams and best-effort loads. With the reservation-based scheduling by L^{4}SCSI, the data streams achieve their requested qualities, the best-effort load is only able to consume the bandwidth not used by the data streams. Without the enforcement of the reservations, the best-effort load interferes with the data streams, causing the data streams to miss the targeted qualities.

Benefits of Statistical Guarantees

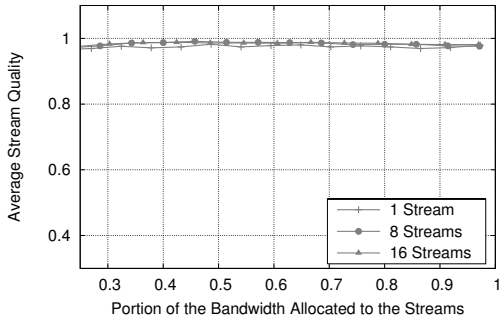
The second motivation of statistical guarantees is to improve the performance of the storage system compared to guarantees based on worst-case assumptions. Using the worst-case execution times presented in Section 6.1, the admission control can accept mandatory data streams with a bandwidth of up to 2.1 MByte/s for the IBM disk and 1.5 MByte/s for the Seagate disk (both using a request size of 64 KByte). Figure 6.7 shows the bandwidths that can be achieved for optional data streams,



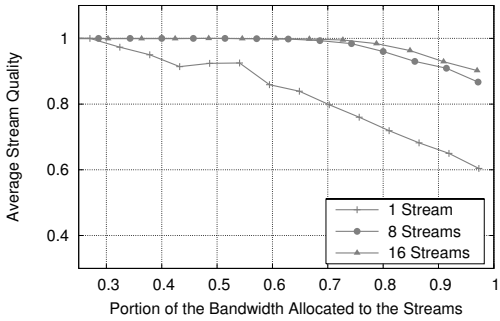
(a) Achieved stream and TIOTEST bandwidths, L⁴SCSI. The stream is read with a guaranteed quality of 0.98, which causes the occasional glitches in the stream bandwidth.



(b) Achieved stream and TIOTEST bandwidths, LINUX. During the execution of the TIOTEST benchmark the bandwidth of the stream drops, as LINUX provides no guarantees for the stream.



(c) Achieved stream qualities, L⁴SCSI. The quality stays at the requested value (0.98), independent of the utilization caused by the streams.



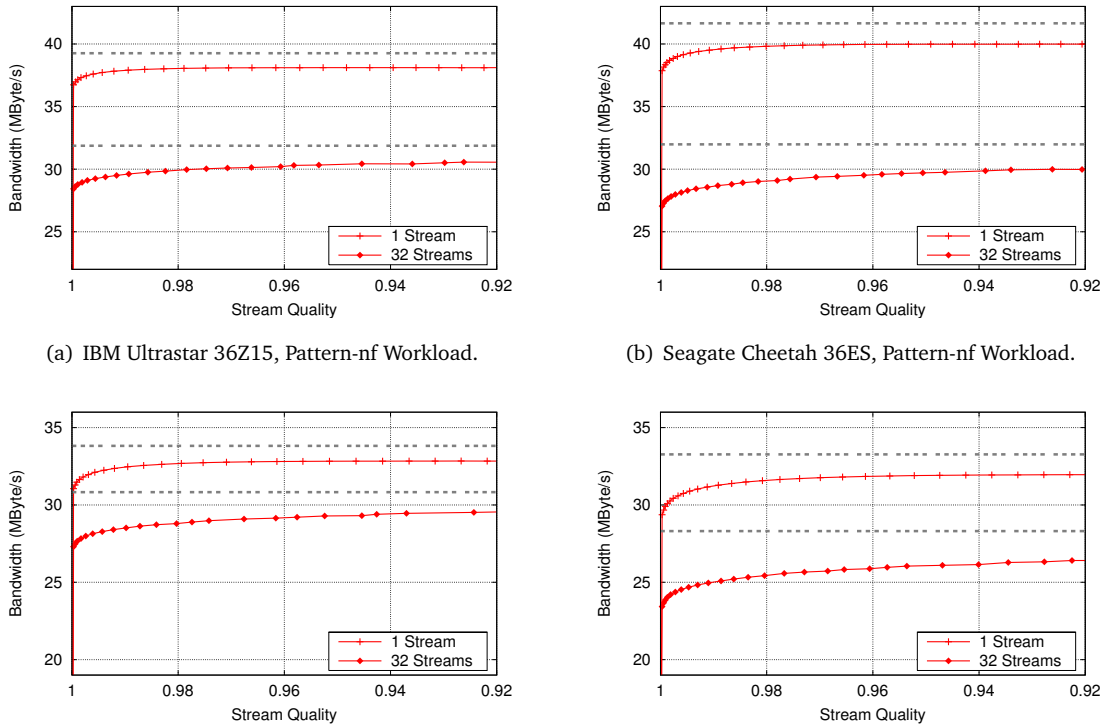
(d) Achieved stream qualities, LINUX. LINUX distributes the available bandwidth to all clients. This results in a drop of the stream qualities, especially if the streams request a large part of the available bandwidth, as the best-effort load is not inhibited from consuming its part of the bandwidth.

Figure 6.6: Data streams and a concurrent best-effort load. The experiments read data streams from the disk and concurrently execute the TIOTEST benchmark creating a best-effort load. With L⁴SCSI, the experiment uses the setup shown in Figure 6.3(b), using the L⁴SCSI test application to read the data streams and executing TIOTEST on L⁴LINUX accessing the disk through a stub driver. With LINUX, the experiment uses the setup shown in Figure 6.3(c), executing LXSTREAM and TIOTEST on native Linux.

depending on the number of data streams used to build up the bandwidth, the requested stream qualities and the workloads of the streams.

The graphs show that not only the stream quality effects the achievable bandwidth, but also the number of streams used to generate the bandwidth. This influence is caused by the enforcement of the guarantees of the data streams, which will be discussed in detail in the next section.

With respect to the dependency on the stream quality, the graphs show that already a slight decrease of the quality of the data streams significantly increases the bandwidth that can be utilized by the data streams. With a stream quality of 0.999, between 85 % and 95 % of the available average-case bandwidth can be utilized for the data streams. However, the stream bandwidth does never reach the average-case bandwidth, a constant gap stays between the stream and average-case bandwidth. This gap is caused by the approximation described in Section 3.2.2 to consider requests that overlap with



(a) IBM Ultrastar 36Z15, Pattern-nf Workload.

(b) Seagate Cheetah 36ES, Pattern-nf Workload.

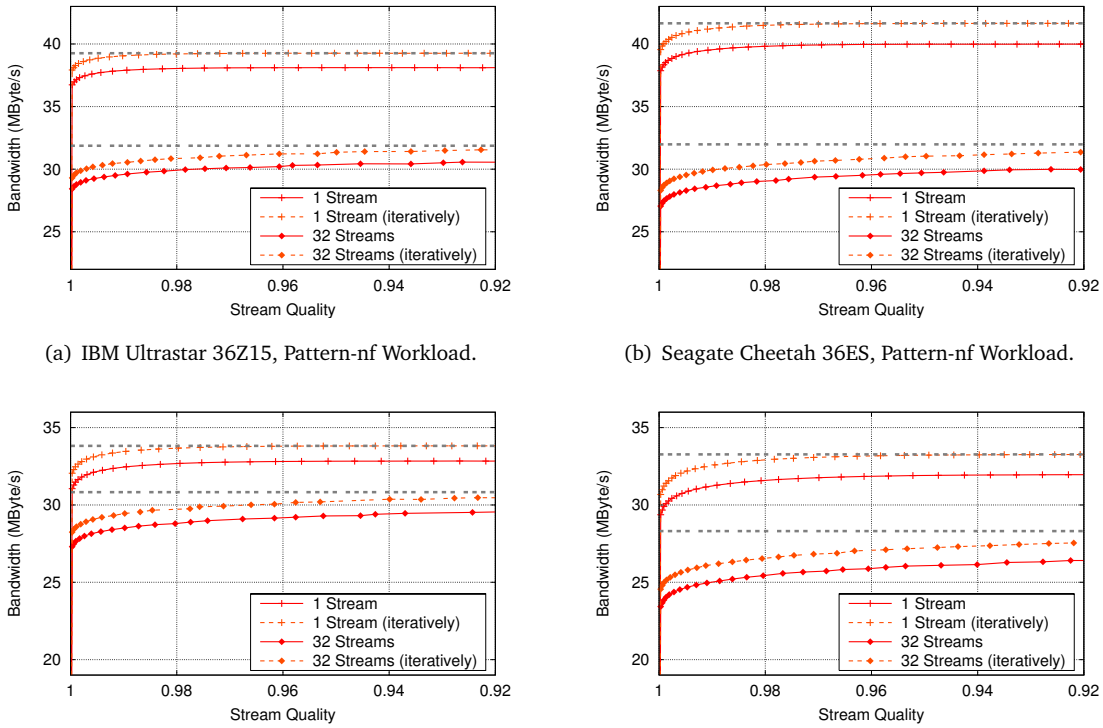
(c) IBM Ultrastar 36Z15, Pattern-hf Workload.

(d) Seagate Cheetah 36ES, Pattern-hf Workload.

Figure 6.7: Maximum bandwidth that can be utilized by data streams depending on the stream quality. The graphs are created determining the maximum bandwidth of the streams that is accepted by the admission control for a given quality. The dashed lines denote the average-case bandwidths, calculated based on the average execution time of the disk requests. The difference of the achievable bandwidths for the various number of streams results from the enforcement of the quality-of-service guarantees, which will be discussed in Section 6.2.3. The achievable bandwidths based on the worst-case execution times are 2.1 MByte/s for the IBM disk and 1.5 MByte/s for the Seagate disk using a period length of 1000 ms and a request size of 64 KByte.

the new period at the end of a period. Reducing the available time for the admission by the worst-case execution time of a request causes the admission control to not fully utilize the disk bandwidth for data streams.

This approximation is necessary to consider the time consumed by requests overlapping from the previous period at the beginning of a new period in the calculation of the reservation time. In the general case, it is not possible to calculate the exact amount of time that is consumed by the overlapping requests. However, in the case of task sets with uniform periods, this time can be calculated. It is defined by the tail of the resulting execution time Z_n that exceeds the period length. Z_n is the random variable describing the aggregated execution time after the last optional stream (i.e., the stream with the lowest priority) is executed. The knowledge of the exact time consumed by the overlapping requests (described by a random variable), can then be use to calculate the reservation time without requiring an approximation. The calculation is done iteratively, starting with zero random variable describing the overlap time in the first iteration. Figure 6.8 includes the results of this iterative approach, showing that now the bandwidth that can be utilized for data streams eventually reaches the average-case bandwidth.



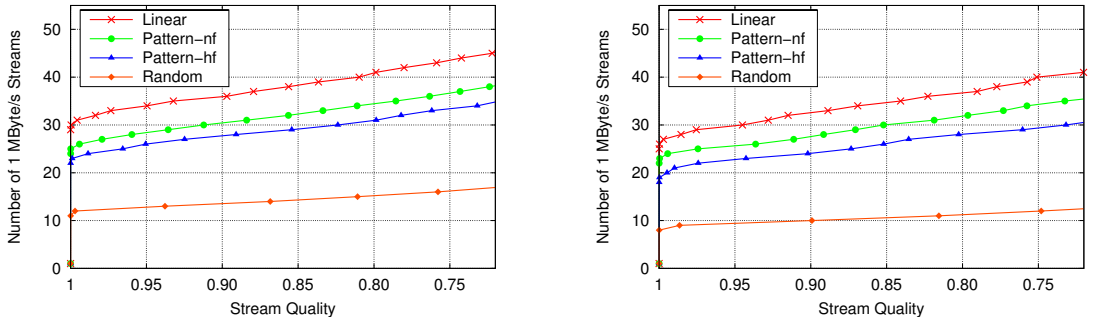
(a) IBM Ultrastar 36Z15, Pattern-nf Workload.

(b) Seagate Cheetah 36ES, Pattern-nf Workload.

(c) IBM Ultrastar 36Z15, Pattern-hf Workload.

(d) Seagate Cheetah 36ES, Pattern-hf Workload.

Figure 6.8: Maximum bandwidth that can be utilized by data streams depending on the stream quality. In addition to the results shown in Figure 6.7, the graphs include the results for the admission control based on the iterative calculation of the reservation times with uniform periods.



(a) IBM Ultrastar 36Z15

(b) Seagate Cheetah 36ES

Figure 6.9: Maximum number of data streams with a bandwidth of 1 MByte/s that are accepted by the admission control depending on the stream quality.

Besides the overall available bandwidth, the number of streams that can be handled is another important property of a storage system. The results presented in Figure 6.8 show that the actual number of streams affect the available bandwidth. Figure 6.9 shows the number of data streams with a bandwidth of 1 MByte/s that can be handled by the disk depending on the requested qualities. The effect of the number of streams on the bandwidth is considered in the calculation of the graphs by using separate distributions for each number of streams to calculate the maximum possible quality.

6.2.3 Costs of Enforcing the Quality-of-Service Guarantees

The results presented in Figure 6.7 and 6.8 already indicate that the enforcement of the quality-of-service guarantees affects the bandwidth the storage system is able to provide, especially for a larger number of data streams. The guarantees of the streams are enforced by the algorithm to create the Dynamic Active Subset (DAS) presented in Section 3.3.1. The algorithm ensures that no streams are added to the subset if the execution of a request of these streams would cause a stream with a higher priority to miss its guarantee.

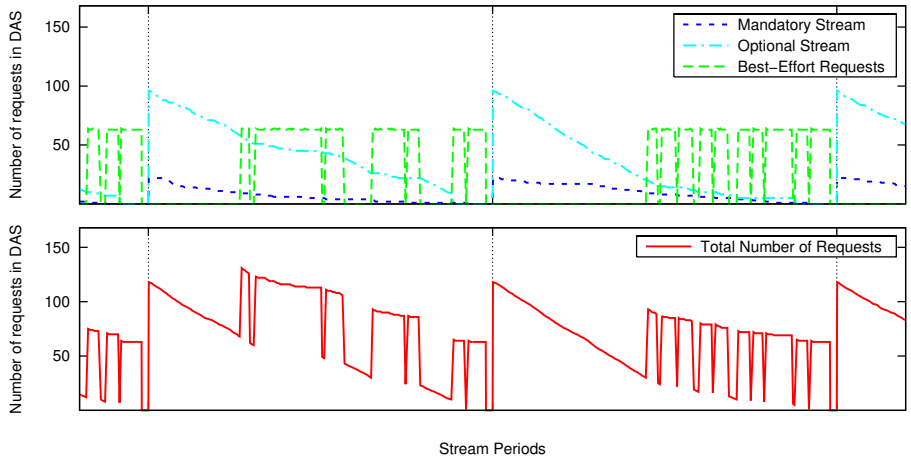
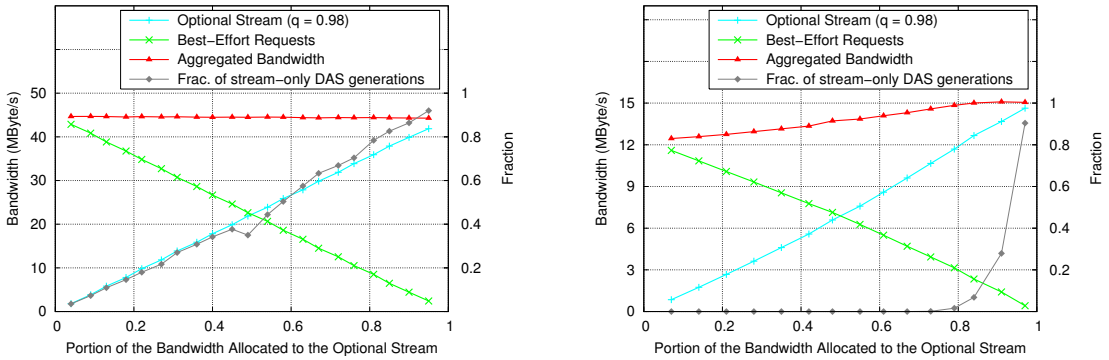


Figure 6.10: Construction of the Dynamic Active Subset (DAS). The setup contains a mandatory stream, an optional stream and a best-effort load. The two streams occupy about 60 % of the disk bandwidth, the best-effort load consists of a queue containing 64 requests at any time. The upper graph shows the number of requests of the streams and the best-effort added to the DAS each time it is created. At the beginning of the periods, only the requests of the two streams are added to the DAS, the best-effort load is only added at times it cannot endanger the guarantees of the streams. The lower graph shows the resulting size of the DAS.

Figure 6.10 shows an example of the construction of the DAS. The requests of the best-effort load (which has the lowest priority) are only added to the subset at times they cannot endanger the guarantees of the streams. This results in DAS instances that only contain a small number of requests, as indicated by the lower graph shown in Figure 6.10. To exclude some of the available requests from the scheduling affects the ability of the scheduler to create an optimal schedule. In particular, the scheduler might not be able to pick the requests that results in the shortest execution time. Instead, the scheduler might be forced to pick a request that results in a long execution time, but is required to be executed to meet a given guarantee.

Figure 6.11 shows the results of the initial experiment performed to analyze the influence of the enforcement of the guarantees on the achievable disk bandwidth. The experiment consists of an optional data stream and a greedy best-effort load simultaneously reading from the disk and measures the achieved aggregated bandwidth. Throughout the experiment, the portion of the bandwidth allocated by the stream is increased, as this increases the frequency of the situations where the scheduler is forced to enforce the stream guarantee. The results indicate no negative effect on the achievable bandwidth. With the linear workload, the achieved bandwidth stays constant throughout the experiment. With the random workload, the bandwidth even increases for a larger stream bandwidth, although the scheduler is forced to enforce the stream guarantees more often, indicated by the in-



(a) Linear Workload. (b) Random Workload.

Figure 6.11: Influence of the DAS scheduling on the achievable bandwidth, IBM Ultrastar 36Z15. The setup used in the experiments consists of an optional data stream with a requested quality of 0.98 and a greedy best-effort load. Throughout the experiment, the portion of the bandwidth used by the stream is increased and the resulting bandwidth of the best-effort load is measured. Additionally, the fraction of DAS generations is measured that only consisted of the stream requests due to the enforcement of the stream guarantee. Figure (a) shows the results of the experiment using the linear workload, Figure (b) the results using the random workload.

creased fraction of DAS generations that only include the stream requests. The increase of the bandwidth results from the larger number of requests available for the scheduling with a higher stream bandwidth, which enables the scheduler to generate a better schedule for the random workload.

The absence of an effect with the linear workload results from the order in that the requests of the stream and the best-effort load are executed. Figure 6.12 shows the influence of the enforcement of the stream guarantees on the execution order of the requests with the linear workload. As the SATF scheduler chooses the requests to execute solely based on the position of the requests, the scheduler stays with executing of either the requests of the stream or the requests of the best-effort load for the linear workload. The scheduler is only forced to abandon the contiguous execution twice during a period, causing only a negligible effect on the bandwidth. First, the scheduler needs to switch to the stream requests to start the execution of the stream, and second it switches back to the best-effort requests once it completed the execution of the stream requests.

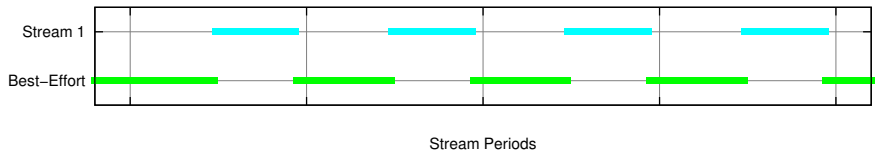
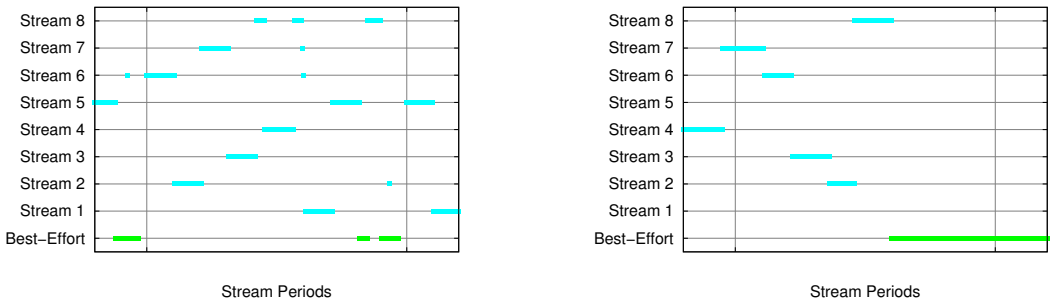


Figure 6.12: Execution order of the stream and best-effort requests for the linear workload. The bars consist of single points, each representing the execution of a request.

This situation changes if the scheduler needs to enforce the guarantees of more than one stream. Figure 6.13 shows the resulting request execution order for a setup consisting of eight streams and a best-effort load, both with enforced guarantees (Fig. 6.13(a)) and without the enforcement of the stream guarantees (Fig. 6.13(b)). To enforce the guarantees of the streams, the scheduler is forced to switch more often between the streams than without the enforcement of the guarantees, causing



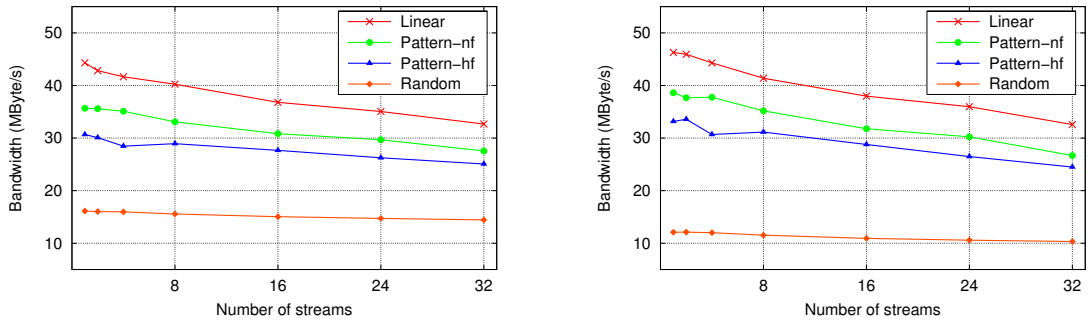
(a) Execution order with enforced stream guarantees.

(b) Execution order without guarantees.

Figure 6.13: Request execution order of eight streams and a best-effort load. Figure (a) shows the execution order with the enforcement of the stream guarantees, Figure (b) shows the execution order without an enforcement of guarantees.

a larger effect on the bandwidth. On the other hand, the enforcement of the guarantees ensures that all streams are executed, whereas without the guarantees a progress is not ensured for the streams, in the example shown in Figure 6.13(b) no requests are executed for the first and fifth stream.

Figure 6.14 shows the achieved bandwidths depending on the number of streams. The results show that the number of streams pose the major cause for the effect on the achievable bandwidth. The maximum effect is caused with the linear workload, using 32 streams the achievable bandwidth drops by 26 % for the IBM disk and 30 % for the Seagate disk compared to the bandwidth achieved using a single stream.

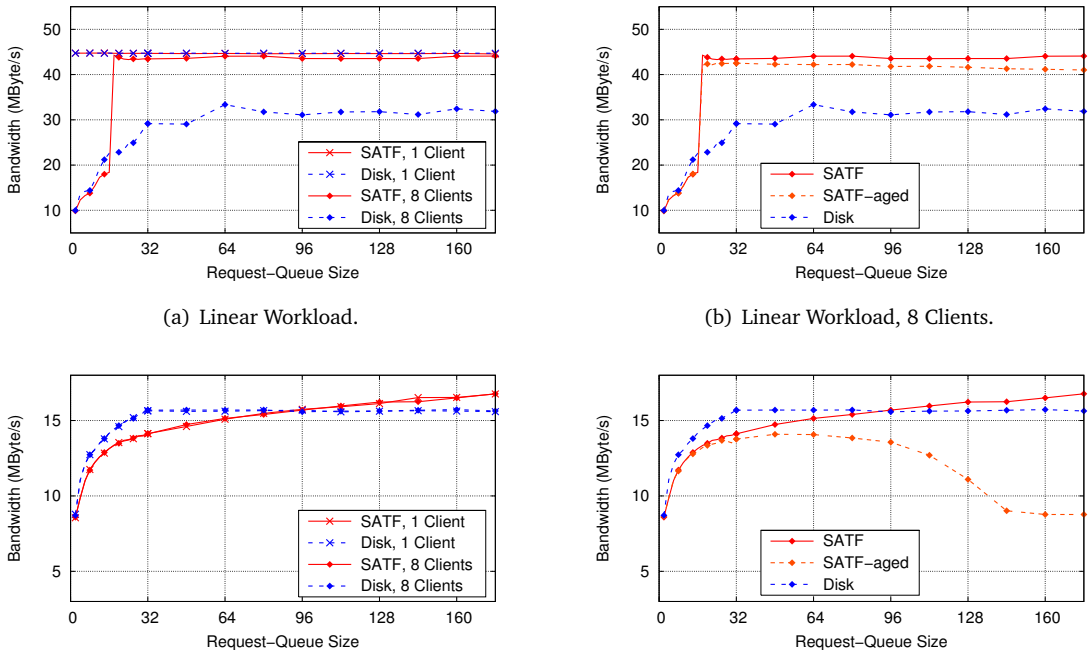


(a) IBM Ultrastar 36Z15.

(b) Seagate Cheetah 36ES.

Figure 6.14: Maximum bandwidth that can be achieved depending on the number of streams. The graphs show the aggregated bandwidths measured using a setup consisting of the respective number of streams with a quality of 0.98, and a best-effort load. The stream setup is created using the maximum bandwidth the admission control accepts, evenly distributed to the streams.

The enforcement of the quality-of-service guarantees also avoids the use of the internal scheduler of the disk drive. Requests cannot be easily canceled once they are issued to the disk drive, which might be necessary to prioritize another request to enforce whose guarantees. Also, the queuing of the requests in the disk drive interferes with the accounting of the request execution times of the requests. Figure 6.15 shows the results of experiments comparing the bandwidths that can be achieved using the disk scheduler and the SATF scheduler used by L^4SCSI. For the random workload (Fig. 6.15(c)), the disk scheduler initially provides a higher bandwidth. However, if the queue-depth



(a) Linear Workload.

(b) Linear Workload, 8 Clients.

(c) Random Workload.

(d) Random Workload, 8 Clients.

Figure 6.15: Comparison of the SATF scheduler and the disk scheduler, IBM Ultrastar 36Z15. The graphs show the aggregated bandwidths achieved running the benchmark with the respective request-queue depths. With the SATF scheduler, the disk scheduler is "disabled" by issuing only one request at a time to the disk. With the disk scheduler, the driver does not order the requests (i.e., uses a FIFO policy). For queue depths larger than the disk is able to handle (32 for both disks), the remaining requests are queued by the driver. Appendix A.3 contains the complete results for both disks and all workloads.

exceeds the number of requests the disk is able to queue internally (which are 32 requests for both the IBM disk and Seagate disk), the disk scheduler cannot benefit from the additional requests, whereas the SATF scheduler can exploit the whole request queue. Therefore, the SATF scheduler is able to provide a higher bandwidth for larger queue sizes.

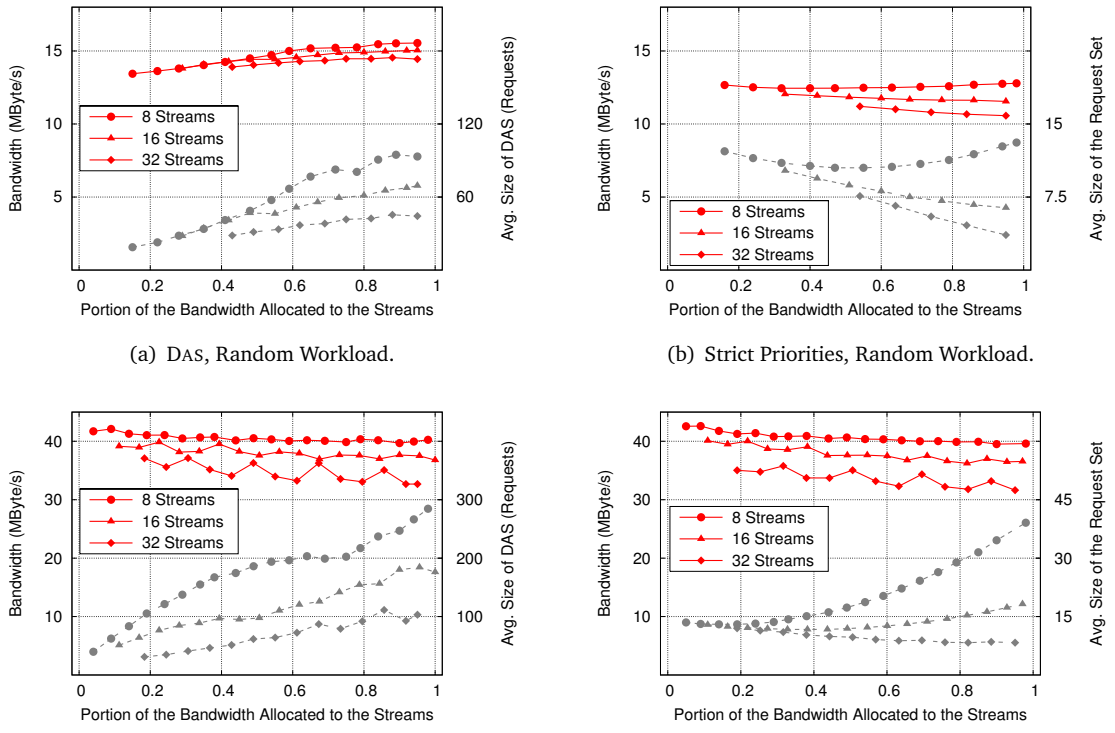
The experiments using the linear workload (Fig. 6.15(a)) provide a different result. With a single client, the disk scheduler and the SATF scheduler achieve similar bandwidths, as the scheduler has no effect on the linear load. With several clients, the bandwidth achieved by the disk scheduler significantly drops, whereas the SATF scheduler eventually achieves the same bandwidth as with a single client. However, this comes at the cost of the starvation of some of the clients, as the SATF scheduler ensures no fairness between its clients, whereas the disk scheduler ensures a progress for all clients. Figure 6.15(b) includes the bandwidths achieved with a slightly modified SATF scheduler, which avoids the starvation of clients by forcibly scheduling requests once they have aged too much (this version of the SATF scheduler is also called *Shortest Access Time First with Urgent Forcing* (SATFUF) [42]). For the linear workload, the modified SATF scheduler achieves a similar bandwidth than the unmodified scheduler. However, the SATFUF scheduler tends to degrade to a FIFO scheduler if the number of aged requests grows too much, as it happens for the random workload shown in Figure 6.15(d). More complex modifications to the SATF scheduler are necessary to both provide fairness to concurrent clients and to optimize the bandwidth [42].

6.3 Benefits of the Dynamic Active Subset

The design of the Dynamic Active Subset (DAS) aims to achieve two goals, first to allow the early inclusion of sporadic real-time requests and best-effort requests without interfering with the guarantees of the data streams; and second to provide the request scheduler with as much requests as possible to improve the performance achieved by the scheduler.

The first of the two goals is achieved by using the slack time of the streams to include additional sporadic real-time requests or best-effort requests as soon as the execution of these requests does not endanger the guarantees of the data streams. This approach provides the flexibility to implement a variety of scheduling policies, for instance to prioritize best-effort requests over stream requests to provide good response times for best-effort requests similar to the ΔL scheduler [11].

The second goal is achieved by adding the requests of as many streams as possible to the DAS (including sporadic real-time and best-effort requests), also using the slack time of the streams to decide whether the requests of streams with a lower priority can be added to the DAS. This enables the request scheduler to pick the next request to execute from a larger set of requests compared to the execution of the streams strictly according to their priorities, which restricts the request scheduler to pick a request only from the stream with the highest priority.



(a) DAS, Random Workload.

(b) Strict Priorities, Random Workload.

(c) DAS, Linear Workload.

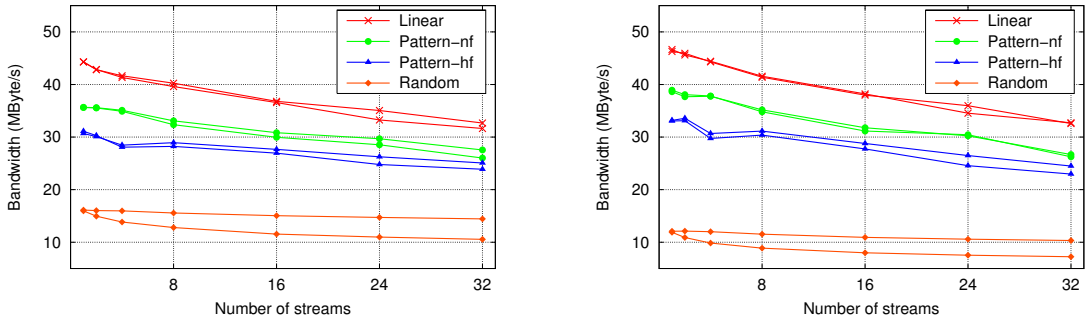
(d) Strict Priorities, Linear Workload.

Figure 6.16: Achieved bandwidths with the DAS scheduling compared to the scheduling strictly using the stream priorities, IBM Ultrastar 36Z15. The experiment executes setups consisting of various streams and a best-effort load that consists of a queue containing 16 requests at any time. The experiment measures the aggregated achieved bandwidth (displayed by the solid graphs) and the average size of the request set available to the request scheduler (displayed by the dashed graphs). Throughout the experiment, the portion of the bandwidth allocated by the streams is increased. This results in a larger number of requests executed by the individual streams.

Figure 6.16 shows the influence of the DAS scheduling on the achievable bandwidth compared to the scheduling strictly following the priorities of the streams. The setups used in the experiments consist of several data streams and a simultaneous best-effort load consisting of a request queue containing 16 requests at any time, all reading from the disk. The experiments measure the aggregated bandwidth achieved by these setups varying the bandwidth allocated to the streams, as this defines the number of requests executed for each data stream. The results of the experiment using the random workload (Fig. 6.16(a)) confirm the initial prediction, as more requests are provided to the request scheduler as a result of the increased stream bandwidth, the scheduler is able to achieve a higher bandwidth.

In contrast, the bandwidth even slightly drops for the experiment executing the streams according to their priorities (Fig. 6.16(b)). This drop is caused by a decrease of the number of requests the scheduler can choose from (denoted by the average size of the request set), especially for a large number of streams. In particular, if the bandwidth allocated by the streams is distributed over a large number of streams, the individual stream only consists of a few requests. If a large portion of the available bandwidth is used by the streams, the request scheduler mostly picks requests from this small number of requests. For a smaller portion of the bandwidth used by the streams, the scheduler more often picks requests from the best-effort load, which provides the scheduler with a constant number of 16 requests in this experiments.

With the linear workload, the number of requests provided to the request scheduler has no effect on the bandwidth achieved by the scheduler. Both the experiments using the DAS (Fig. 6.16(c)) and scheduling the requests according to their priorities (Fig. 6.16(d)) achieve the nearly the same bandwidths, although using the DAS provides the request scheduler with a significantly larger number of requests. This result is caused by the ineffectiveness of the request scheduler with the linear workload.



(a) IBM Ultrastar 36Z15.

(b) Seagate Cheetah 36ES.

Figure 6.17: Maximum bandwidths achieved with the DAS scheduling compared to the scheduling strictly using the stream priorities. The graphs repeat the results shown in Figure 6.15 (the upper graphs in the respective graph pairs), together with the results obtained for the experiment scheduling the streams strictly according to their priorities (the lower graphs). Table 6.9 in detail lists the differences in the achieved bandwidths.

The results of the experiments shown in Figure 6.16 indicate that the effectiveness of the DAS with respect to the achieved bandwidth strongly depends on the workload. Figure 6.17 compares the achievable bandwidths for all workloads, both with and without using the DAS. The graphs include the results already presented in the previous section in Figure 6.14, together with the results of the same experiment repeated, but scheduling the requests strictly based on their priorities. Table 6.9 summarizes the gain in bandwidth achieved using the DAS. With the random workload, the achieved

Workload	IBM Ultrastar 36Z15 Bandwidth (KByte/s) w/o DAS	IBM Ultrastar 36Z15 Bandwidth (KByte/s) DAS	IBM Ultrastar 36Z15 Gain (%)	Seagate Cheetah 36ES Bandwidth (KByte/s) w/o DAS	Seagate Cheetah 36ES Bandwidth (KByte/s) DAS	Seagate Cheetah 36ES Gain (%)
Linear, 8 Streams	40559	41216	1.6	42365	42567	0.5
Pattern-nf, 8 Streams	33125	33879	2.3	35633	36022	1.1
Pattern-hf, 8 Streams	28908	29625	2.5	31101	31876	2.5
Random, 8 Streams	13095	15932	21.7	9085	11808	30.0
Linear, 16 Streams	37435	37677	0.6	38885	39125	0.6
Pattern-nf, 16 Streams	30681	31572	2.9	31890	32535	2.0
Pattern-hf, 16 Streams	27582	28319	2.7	28413	29476	3.7
Random, 16 Streams	11823	15408	30.3	8183	11180	36.6
Linear, 32 Streams	32388	33458	3.3	33355	33494	0.4
Pattern-nf, 32 Streams	26640	28215	5.9	26885	27312	1.6
Pattern-hf, 32 Streams	24461	25669	4.9	23521	25084	6.6
Random, 32 Streams	10799	14786	36.9	7418	10581	42.6

Table 6.9: Differences of the achieved bandwidths using the DAS scheduling compared to the scheduling strictly using the stream priorities.

bandwidth is increased by up to 43 %, whereas the gain using the other workloads is significantly lower.

6.4 Integration into the Overall System Architecture

The evaluation presented so far in this chapter provided an isolated view on the properties of the disk scheduling. The following section analyzes the effects of the integration of the storage system into the overall DROPS architecture, in particular the costs of connecting L⁴SCSI to L⁴LINUX and the CPU-time demand of L⁴SCSI.

6.4.1 Connecting L⁴SCSI and L⁴LINUX

L⁴LINUX provides the main environment for best-effort applications in DROPS. With L⁴SCSI running as a separate resource manager, disk requests of best-effort applications need to be forwarded to L⁴SCSI using a stub driver in L⁴LINUX. The use of a stub driver to forward requests to L⁴SCSI causes an additional overhead on the execution of the disk request compared to the execution of the requests directly by L⁴LINUX. Table 6.10 shows the results of experiments analyzing this overhead by running the TIOTEST benchmark [3] on L⁴LINUX using the stub driver and comparing the achieved performance with the results obtained running TIOTEST on L⁴LINUX and native LINUX both using the internal SCSI driver to access the disks.

	Bandwidth (KByte/s)			CPU Utilization (%)		
Test	LINUX	L⁴LINUX	L⁴LINUX + L⁴SCSI	LINUX	L⁴LINUX	L⁴LINUX + L⁴SCSI
4 KByte Requests						
Linear Read	45 302	45 901	45 437	28.64	35.34	36.47
Random Read	1 262	1 125	1 135	1.83	2.33	3.01
Linear Write	43 589	43 024	43 121	45.75	52.36	51.16
Random Write	2 038	2 033	2 763	6.25	6.93	10.43
64 KByte Requests						
Linear Read	46 514	46 514	46 464	30.23	31.21	33.40
Random Read	13 649	12 101	12 059	8.71	8.80	9.92
Linear Write	43 733	42 995	43 071	44.32	45.24	44.35
Random Write	16 610	16 168	19 179	16.52	16.84	20.58

(a) IBM Ultrastar 36Z15.

	Bandwidth (KByte/s)			CPU Utilization (%)		
Test	LINUX	L⁴LINUX	L⁴LINUX + L⁴SCSI	LINUX	L⁴LINUX	L⁴LINUX + L⁴SCSI
4 KByte Requests						
Linear Read	51 088	50 955	50 233	33.98	41.84	41.37
Random Read	850	751	765	1.29	1.62	2.07
Linear Write	45 105	44 285	44 033	47.29	53.46	51.03
Random Write	1 796	1 785	2 217	5.52	6.16	8.20
64 KByte Requests						
Linear Read	51 145	51 053	50 652	33.18	35.07	37.28
Random Read	10 513	9 142	9 307	6.29	6.52	7.79
Linear Write	45 359	44 348	44 416	46.06	46.69	44.77
Random Write	13 818	13 176	14 223	13.78	13.79	15.29

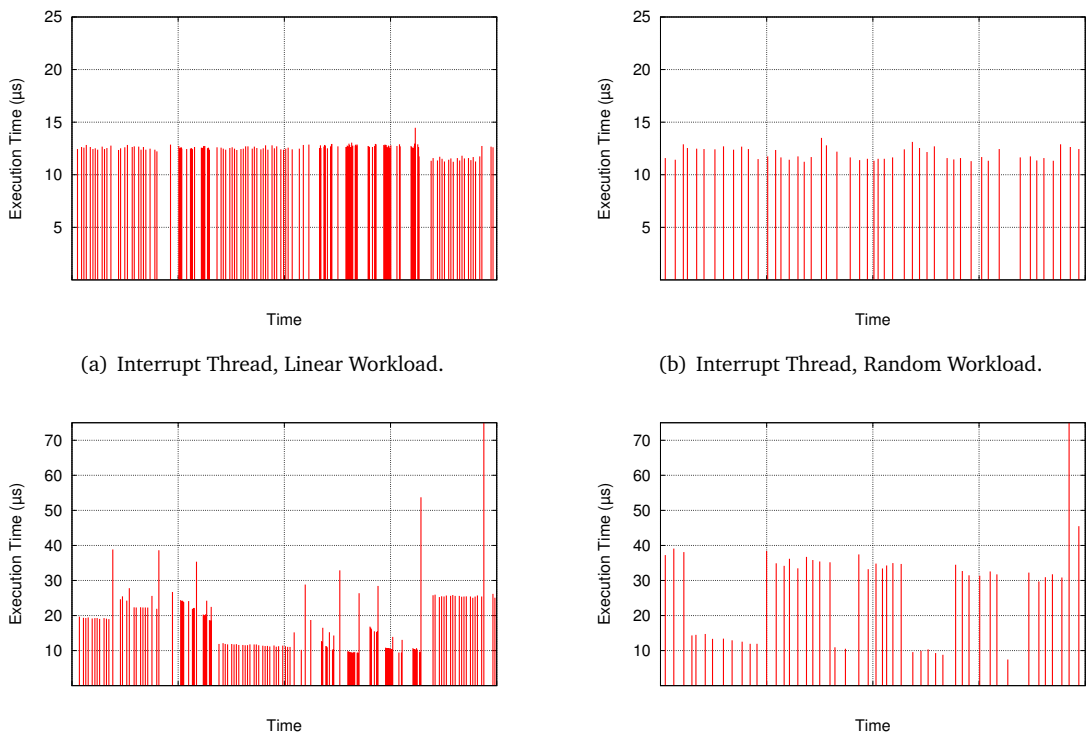
(b) Seagate Cheetah 36ES.

Table 6.10: TIOTEST Results. The experiments compare the performance of L⁴SCSI connected to L⁴LINUX with the results achieved with L⁴LINUX and native LINUX both using the internal disk driver. The tables show the achieved bandwidths and the generated CPU utilization for the various tests performed by TIOTEST. The CPU utilization is measured using the performance counters of the Pentium 3 processor.

The results show no clear effect of using the stub driver on the achieved performance of TIOTEST. For most of the performed tests, L⁴LINUX using L⁴SCSI achieves a similar bandwidth compared to L⁴LINUX using the internal disk driver to access the disks. The overhead induced by the additional communication between L⁴LINUX and L⁴SCSI causes only a slight increase of the CPU utilization. With the random-write test, L⁴LINUX achieves even a higher bandwidth using L⁴SCSI than with the internal driver, although causing a more noticeable increase of the CPU utilization. This result might be caused by the different queueing behavior in L⁴LINUX using L⁴SCSI, as the stub driver immediately removes requests from the request queue in L⁴LINUX and forwards them to L⁴SCSI.

6.4.2 CPU-Time Demand of L⁴SCSI

L⁴SCSI requires sufficient CPU time to successfully enforce the quality-of-service guarantees given for the execution of the data streams. In particular, the driver must be able to timely issue requests to the disk to avoid the disk to stall, which could result in the violation of a guarantee as the admission control assumes the driver to fully utilize the disks. With L⁴SCSI, two threads are involved in the scheduling of the disk requests, a thread handling the interrupts of the SCSI host adapter and the thread performing the actual request scheduling. Both threads demand guarantees on the available CPU time, which must be included in the scheduling analysis of the CPU scheduler.



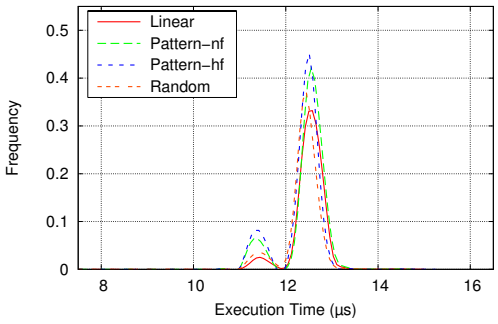
(a) Interrupt Thread, Linear Workload.

(b) Interrupt Thread, Random Workload.

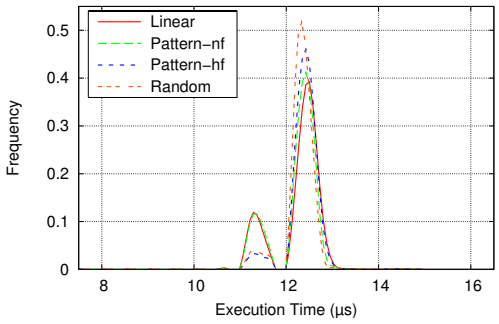
(c) Request-Handling Thread, Linear Workload.

(d) Request-Handling Thread, Random Workload.

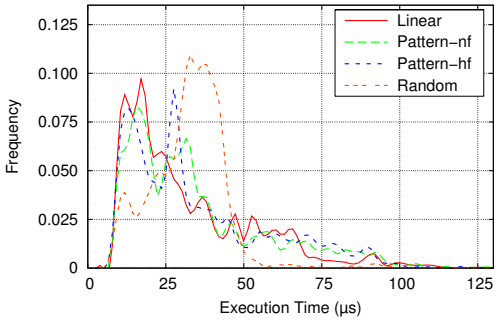
Figure 6.18: Activations of the Interrupt and Request-Handling threads using the stream setups described in Appendix A.1. The graphs show a trace of 200 ms of the execution of the threads. Each line represents a single activation of a thread. The position of the line on the x-axes denotes the time of the activation, the height of the line the CPU time consumed during this activation.



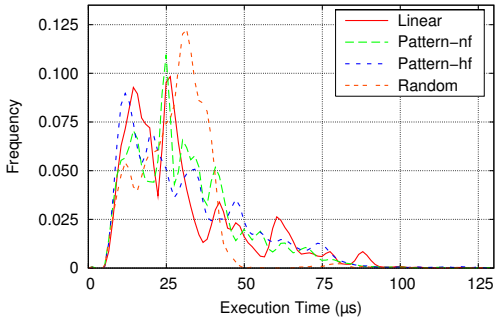
(a) IBM Ultrastar 36Z15, Interrupt Threads.



(b) Seagate Cheetah 36ES, Interrupt Threads.



(c) IBM Ultrastar 36Z15, Request-Handling Threads.



(d) Seagate Cheetah 36ES, Request-Handling Threads.

	Interrupt Thread				Request-Handling Thread			
Workload	min	max	avg	dev	min	max	avg	dev
Linear	10.94	14.94	12.50	0.34	8.16	265.12	33.05	22.94
Pattern-nf	10.94	14.72	12.45	0.44	7.81	252.26	35.26	23.44
Pattern-hf	10.56	15.10	12.34	0.44	7.78	253.38	35.40	23.68
Random	11.04	14.56	12.37	0.36	6.85	221.63	32.85	15.61

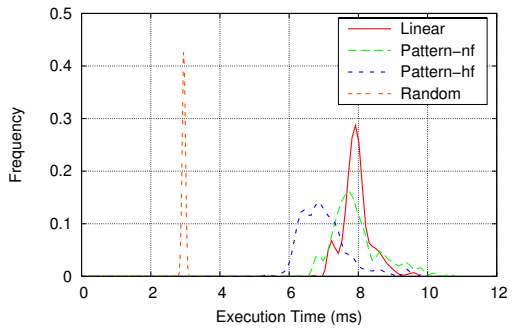
(e) Execution times (in µs), IBM Ultrastar 36Z15.

	Interrupt Thread				Request-Handling Thread			
Workload	min	max	avg	dev	min	max	avg	dev
Linear	10.94	14.94	12.26	0.49	7.90	197.12	31.23	20.35
Pattern-nf	9.60	14.78	12.22	0.47	7.71	172.96	31.45	17.91
Pattern-hf	11.07	14.91	12.38	0.32	7.39	186.69	31.76	20.05
Random	10.66	14.59	12.31	0.32	6.21	187.55	28.44	14.53

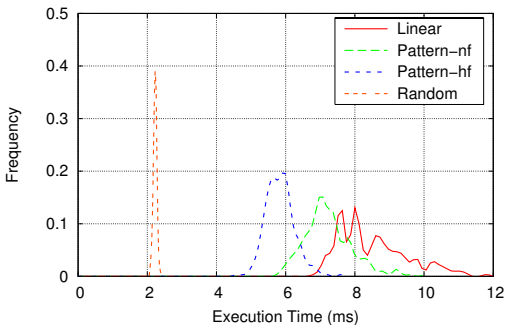
(f) Execution times (in µs), Seagate Cheetah 36ES.

Figure 6.19: CPU-time demand of the Interrupt and Request-Handling threads. The graphs show the distributions of the execution times of the threads for a single activation, executing the stream setups described in Appendix A.1.

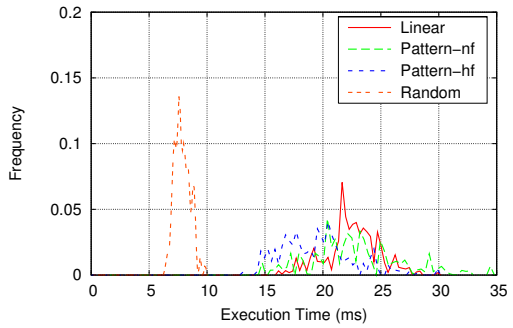
To allow the CPU scheduler to incorporate the demands of the two threads into the scheduling analysis, the exact specification of the CPU demand must be known. Figure 6.18 shows traces of the execution of both threads, illustrating both the release times of the threads and the respective execution times. The release times of the threads provide an aperiodic behavior, following the execution times of the disk requests. The execution time of the interrupt thread is nearly constant. The execution time of the request-handling thread shows a larger variation, which is caused by the varying number of requests the request scheduler handles, the actual number is particularly influenced by the creation of the DAS. Figure 6.19 summarizes the characteristics of the execution times required by



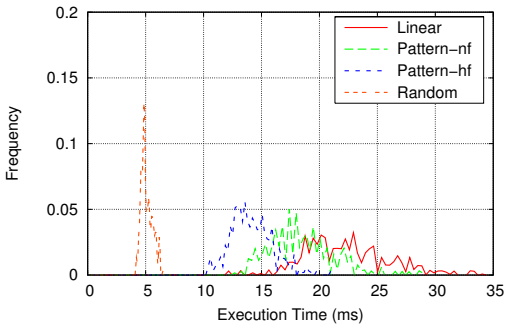
(a) IBM Ultrastar 36Z15, Interrupt Threads.



(b) Seagate Cheetah 36ES, Interrupt Threads.



(c) IBM Ultrastar 36Z15, Request-Handling Threads.



(d) Seagate Cheetah 36ES, Request-Handling Threads.

	Interrupt Thread				Request-Handling Thread			
Workload	min	max	avg	dev	min	max	avg	dev
Linear	7.08	9.66	7.95	0.42	16.27	30.05	22.48	2.29
Pattern-nf	6.68	10.62	7.95	0.72	14.14	34.62	22.73	3.83
Pattern-hf	5.48	9.71	6.99	0.67	13.20	30.08	19.45	3.02
Random	2.85	3.09	2.96	0.04	6.47	9.88	7.85	0.66

(e) CPU usage (in ms), IBM Ultrastar 36Z15.

	Interrupt Thread				Request-Handling Thread			
Workload	min	max	avg	dev	min	max	avg	dev
Linear	6.90	11.82	8.51	1.00	12.22	34.23	22.26	3.63
Pattern-nf	5.77	9.97	7.24	0.70	12.70	29.04	18.43	2.80
Pattern-hf	4.63	7.64	5.84	0.42	10.41	20.95	14.17	1.79
Random	2.08	2.39	2.22	0.05	4.20	6.46	5.10	0.49

(f) CPU usage (in ms), Seagate Cheetah 36ES.

Figure 6.20: CPU usage of the Interrupt and Request-Handling threads within a replenish interval (1000 ms). The graphs show the distributions of the CPU usage executing the stream setups described in Appendix A.1. The differences in the execution times for the various workloads mainly result from the different numbers of requests executed for the workloads during the replenish interval. The least requests are executed for the random workload as a result of the long request-execution times for the random workload. The most requests are executed for the linear workload, resulting in the highest CPU demand.

L^{4}SCSI. With the implementation of L^{4}SCSI used throughout the evaluation, the threads are provided with sufficient CPU time implementing a model similar to *Deferred Servers* [50], using the real-time extensions of the FIASCO kernel to provide the threads with a time budget that is periodically replenished. Figure 6.20 shows the time required by the threads using a replenish interval of 1000 ms. The

actual budget of the threads is derived using the maximum time required by the threads, resulting in a CPU utilization caused by the threads of about 5 %.

In addition to the execution time required by the threads, the scheduling analysis also needs to account the deadlines that must be met by the threads. With the request scheduling required to avoid the stalling of the disk, the threads are required to immediately respond to an interrupt indicating the completion of a request. This sets the deadline for the execution of the threads to the actual execution time of the thread, posing a tough constraint on the CPU scheduling. This requirement can be relaxed by overlapping the execution of a request with performing the scheduling of the subsequent request, deploying the ability of disks to queue requests.[1] This sets the deadline to the latest time the request needs to be issued to the disk, which is defined by the shortest execution time of a disk request.

[1] With queueing just one request, L^{4}SCSI maintains the control over the execution order of the requests and the accounting, as the disk is still not able to reorder the requests.

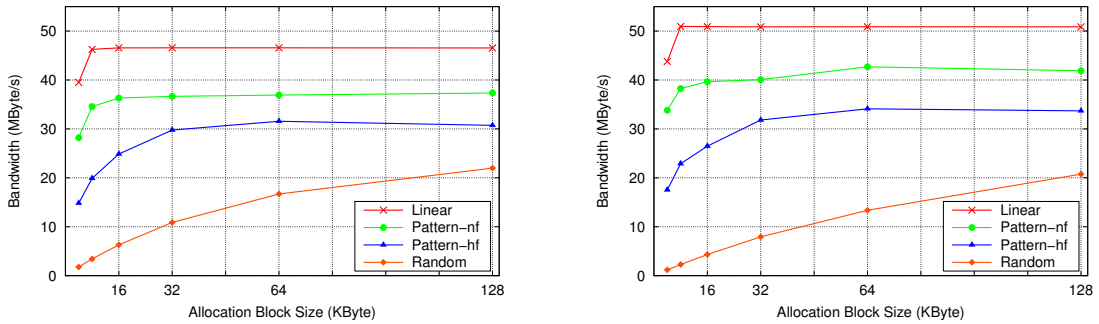
6.5 File System Supporting Quality-of-Service Guarantees

This section evaluates the main two properties of the file-system design presented in Chapter 4, which are first the support of the block allocation using various block sizes and second the streaming client interface.

6.5.1 File Allocation Using Various Block Sizes

Being able to allocate files using various block sizes achieves two goals. On the one hand, it ensures that data streams can be processed using a sufficiently large request size to provide adequate bandwidths. On the other hand, it avoids the wasting of disk space storing small files.

The influence of the request size on the achievable bandwidth is a result of the processing of a disk requests including both operations actually reading or writing data and operations not processing any data. With a small request size, most of the execution time of a disk request is caused by operations not transferring data, in particular the positioning of the disk head. Increasing the request size results in a larger portion of the request-execution time being used to transfer data, as the positioning time is not effected by the request size. Figure 6.21 illustrates this influence on the achievable bandwidth. With the random workload, increasing the block size from 4 KByte to 128 KByte results in an increase of the achieved bandwidth by more than 12 times for the IBM disk and by more than 17 times for the Seagate disk. With the linear workload, the effect is significantly lower as this workload eliminates the overhead caused by the positioning time.



(a) IBM Ultrastar 36Z15.

(b) Seagate Cheetah 36ES.

Figure 6.21: Influence of the file block size. The graphs show the maximum bandwidth achieved contiguously reading files from the disks depending on the block size used to store the files.

6.5.2 Memory Requirements of the Streaming Interface

The streaming client interface and the resulting request execution model presented in Section 4.3 requires sufficient memory to be available to hold the data of the streams. The required amount of memory is influenced by two parameters, the memory requirement imposed by the periodic request scheduling and the consumption of the data by the client.

The periodic execution model of the disk requests requires the requests executed within a period to be available at the beginning of the period, requiring the memory necessary to hold the data processed by the streams also to be available at the beginning of the period. In the best case, the client is able

to process the data within the time defined by a disk period, allowing the file system to implement a double-buffering scheme. This results in the minimum memory demand to be defined by the amount of memory required to hold the data processed within two disk periods. The required amount of memory can increase if the file-system clients demand the stream data to be available over a longer period of time, for instance to decode frames of a video with inter-frame dependencies.

With the data streams handled by the file system using constant bandwidths, the required memory also depends on the buffer required to compensate the varying bandwidth requirements of streams with variable bit rates (VBR), commonly used with encoded video and audio streams. The buffering must ensure that the client of the file system does never stall if it access a file, meaning that the buffer always contains enough data if the client reads a file and that enough free space is available in the buffer if the client writes a file.

The required buffer size can be calculated using data fill and removal functions describing the amount of data that are added to the buffer and removed from the buffer [88]. Figure 6.22 shows a small section of these functions, obtained for the delivery of an MPEG-2 video. The fill function $F(t)$ shows the amount of data added to the buffer at time t, the removal function $R(t)$ the amount of data removed from the buffer at time t.

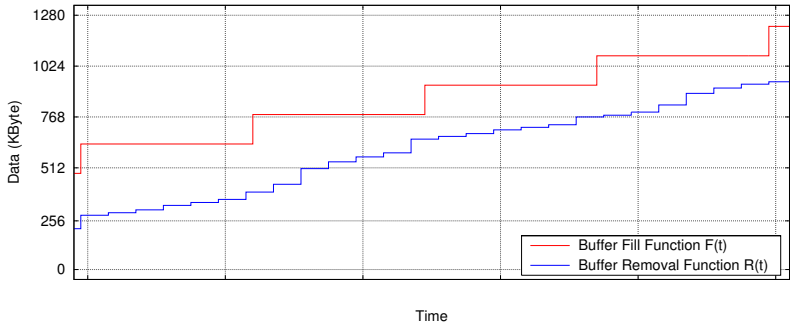


Figure 6.22: Buffer fill and removal functions reading a VBR video. The fill function $F(t)$ shows the periodic adding of a constant amount of data to the buffer. The removal function $R(t)$ also removes the data periodically from the buffer, but the amount of data depends of the encoding of the frames (the shown function is based on the frame-by-frame removal of the data from the buffer).

To prevent the client from stalling, the removal function must always stay below the fill function, meaning that at no time more data is removed from the buffer than filled into the buffer. To achieve this goal, the fill function might be required to be started ahead of the removal function, resulting in a sufficient amount of data to be contained in the buffer at the time the removal of the data starts. More formally, the minimum lead time V_{min} is defined by

$$V_{min} = min(V|F(t+V) - R(t) > 0), \qquad t \geq 0, \tag{6.5}$$

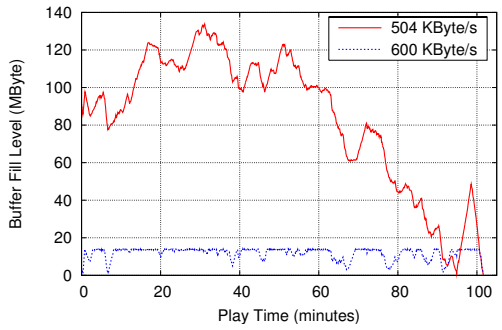
and the function $B(t)$ describing the amount of data contained in the buffer is defined by

$$B(t) = F(t+V_{min}) - R(t). \tag{6.6}$$

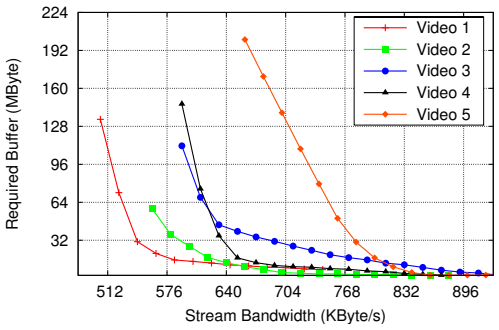
With this definition of the buffer fill function, the buffer size S and with it the amount of memory that must be allocated by the streaming interface is defined by the maximum value reached by the fill function:

$$S = max(B(t)), \qquad t \geq -V_{min}. \tag{6.7}$$

Apparently, the fill function should deliver the data at least at the average rate of the removal function.[2] However, this still can result in a large buffer requirement, as the buffer must hold enough data to compensate occasional large bandwidth requirements of the removal function. The upper graph in Figure 6.23(a) shows the buffer fill function obtained for the delivery of a MPEG-2 video using the average rate of the video to fill the buffer.



(a) Buffer fill function $B(t)$ reading a VBR video (Video 1) using different stream bandwidths (i.e., fill functions $F(t)$) to fill the buffer. The upper graph shows the buffer requirement using the average bandwidth of the video to read the stream, the lower using a larger bandwidth.



(b) Buffer Requirements of VBR videos depending on the stream bandwidth. All graphs start with the stream bandwidth set to the average bandwidth of the videos.

Figure 6.23: Buffer requirements of VBR videos. Figure (a) shows the buffer function $B(t)$ for the delivery of an MPEG-2 video using two different bandwidths to read the file from the disk. Figure (b) shows the buffer requirements of several videos depending on the streams bandwidth. All videos are DVD-quality MPEG-2 videos, representing different video characteristics including black-and-white movies and cartoons.

To alleviate the buffer requirements, the file system can add the data to the buffer at a higher rate, reducing the amount of memory required to cope with the occasional high bandwidth requirements. The lower graph in Figure 6.23(a) shows the fill function obtained adding the data with the rate increased by about 20 %, resulting in a significantly lower buffer size.

However, to effectively benefit from this approach, the file system must avoid that the increased fill rate causes the buffer size to grow beyond the size required to compensate the removal rate. Therefore, the fill function $F'(t)$ used to finally calculate the buffer fill function is defined by

$$F'(t_n) = \begin{cases} F'(t_{n-1}) + b & \text{if} \quad B(t_n) < S \\ F'(t_{n-1}) & \text{otherwise,} \end{cases} \tag{6.8}$$

which means that the file system only adds data to the buffer at time t_n (with b denoting the amount of data added to the buffer) as long as it does not exceed the buffer size S. The minimum buffer size S_{min} is calculated iteratively, finding the smallest buffer size required to ensure that the buffer always contains data:

$$S_{min} = min(S | F'(t + V_{min}) - R(t) > 0) \qquad t \geq -V_{min}. \tag{6.9}$$

Being able to deliver the data at different rates allows a tradeoff between the stream bandwidth and the memory requirement. Figure 6.23(b) shows the memory requirement of a couple of MPEG-2

[2] Of course the fill function can add the data at a lower rate, but this would result in a large lead time and buffer requirement to ensure enough data is stored in the buffer to compensate the lower rate.

videos depending on the bandwidth used to read the file from the disk. The results show that a moderate increase of the stream bandwidth can significantly decrease the memory required to buffer the stream.

6.6 Summary

This section presented an evaluation of the properties of the disk scheduling and file system supporting statistical quality-of-service guarantees. In particular, the accuracy, the benefits and the costs of disk scheduling were analyzed using various workloads and application scenarios.

In summary, the main results obtained throughout the evaluation are:

- The disk request scheduling is able to meet the predictions of the admission model with an error of 5 % for the predictions based on the request-execution-time distributions exactly representing the workload.
- The reservation-based disk scheduling provides an isolation both between individual real-time streams and between real-time streams and simultaneous best-effort loads.
- The use of statistical quality-of-service guarantees significantly increases the portion of the available bandwidth that can be utilize by the admission control to provide guarantees for data streams.
- The costs of enforcing the guarantees of streams mainly depends on the number of streams handled by the system. This influence can cause the available bandwidth to drop by up to 30 % with 32 streams using a linear workload.
- The use of the Dynamic Active Subset (DAS) can increase the available bandwidth by up to 43 % for random workload compared to the execution of the streams strictly according to their priorities. However, the effectiveness largely depends on the workloads of the streams.
- The integration of the storage system into the overall DROPS architecture causes only a negligible overhead.
- The memory required to compensate the varying requirements of VBR streams can be significantly reduced by processing the streams with a moderately higher bandwidth than the average bandwidth of the streams.

The presented results were obtained using two different disks. The results of additional experiments using a series of other disks provide the confidence that the results represent the typical behavior of current disk drives.

Chapter 7

Conclusions and Future Work

This chapter concludes this thesis by summarizing the contributions of the thesis and providing suggestions for future work.

7.1 Contributions

This thesis addressed the challenges raised by providing quality-of-service guarantees in disk-storage systems. The three main contributions of this thesis are:

1. the Quality-Assuring Disk Scheduling,
2. the disk-request scheduling using the Dynamic Active Subset, and
3. the design of a file system that complies with the requirements posed by the enforcement of quality-of-service guarantees as well as that supports the requirements of various file types.

The developed solutions combine the needs of modern disk-storage systems, in particular the ability to efficiently support a variety of different quality-of-service types.

Quality-Assuring Disk Scheduling (Section 3.2)

The Quality-Assuring Disk Scheduling applies the general concepts of the Quality-Assuring Scheduling to the management of disk resources. It provides an admission model that is able to ensure both hard real-time guarantees based on worst-case execution times of disk requests and statistical real-time guarantees based on an admission test using random variables to describe the execution times of disk requests. The admission criterion of the Quality-Assuring Disk Scheduling comprises two main properties:

1. It calculates the required resource demands of data streams such that each stream achieves its quality-of-service guarantees (i.e., a minimum percentage of successfully executed disk requests). This resource demand, called reservation time, provides the disk-request scheduler with sufficient information to enforce the guarantees, in particular with the presence of unpredictable workloads such as sporadic real-time requests or requests of best-effort applications.

2. The admission criterion aims to exactly model the real scheduling, using random variables to describe the execution time of an individual disk request as well as to describe the overall workload of the system. With this approach, the admission model is able to respect situations where the execution of requests does not fully consume the assigned resources. In particular, the admission model can utilize the time not consumed by the execution of requests admitted using worst-case assumptions (i.e., requests with hard real-time guarantees) to admit requests with statistical guarantees.

With these two properties, the Quality-Assuring Disk Scheduling is able to fulfill the individual quality requirements of the clients of the storage system as well as to fully utilize the abilities of the disk drive.

The Dynamic Active Subset (Section 3.3)

The use of the Dynamic Active Subset (DAS) allows the disk-request scheduler to clearly separate the enforcement of real-time guarantees (i.e., the reservation times calculated by the admission control) from the task to optimize the order of the execution of the disk requests. With the DAS, each time the request scheduler needs to choose a request a subset of the outstanding requests is created such that no real-time guarantee can get violated. The actual request scheduling policy is constrained to pick a request from this subset, releasing the policy from the task to consider the enforcement of the real-time guarantees.

The creation of the DAS utilizes the slack time of streams to include as many requests as possible to the subset, which provides a flexible approach to include the execution of both sporadic real-time requests and best-effort requests with the execution of stream requests.

File Systems with Quality-of-Service Guarantees (Chapter 4)

The presented file-system design addresses the requirements posed by the enforcement of quality-of-service guarantees with the disk scheduling. The design comprises of two main elements. First, a flexible block allocation policy is used to provide various block sizes by partitioning the disk into several allocation groups, each providing a fixed block size. This ensures that on the one hand data streams can be processed with an appropriate block size that conforms with the data-stream model, and that on the other hand the fragmentation of the file system is kept low. Second, the file system provides a streaming client interface that matches the data-stream model used with the disk-scheduling.

To evaluate the proposed designs, they were implemented with the DROPS Disk Storage System. The evaluation of this implementation presented in Chapter 6 provides sufficient evidence of the feasibility of the designs, including a discussion of both the benefits and the costs of the proposed solutions.

7.2 Future Work

This thesis opens up various opportunities for future work.

First, the presented designs focus on a single disk. To apply the ideas to multiple disks (e.g., to increase the bandwidth and storage capacity of the system), one could apply the concept of *coarse-grained* striping [98]. With coarse-grained striping, the files are distributed to the disks using large block sizes. This should allow to derive the bandwidth requirements for each individual disks based of the distribution policy, and to apply the admission model and request scheduling presented in this thesis separately to each disk. The file system would need to combine the streams of the individual disks to provide applications with an aggregated stream.

Second, the quality guarantees provided by the admission control are based on the percentage of successfully executed disk requests. However, the final quality provided by an application to its users requires the mapping of this stream quality to the quality notion used by the application, for instance the rate of lost frames for a video player. The influence of the stream quality on the application quality depends on the high-level structure of the stream, for instance how many frames are stored within the disk blocks processed by a single disk request. To describe this dependency, one could use random variables similar to [89].

Finally, the presented support for sporadic real-time requests is limited to an acceptance test that utilizes the disk bandwidth not consumed by data streams. To provide a more comprehensive support, including bandwidth guarantees for sporadic real-time requests, one could further investigate the application of well-known approaches to support sporadic and aperiodic task in real-time systems, such as deferred servers [50] or constant-bandwidth servers [4].

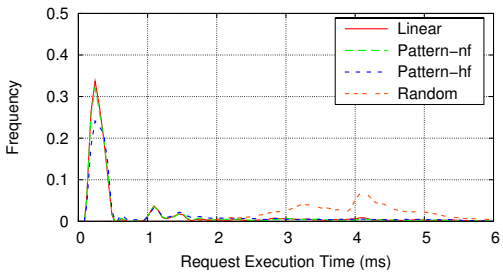
Appendix A

Measurement Results

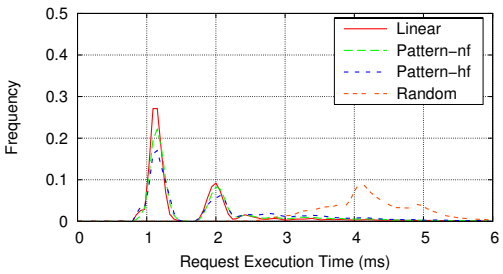
A.1 Achieved Stream Qualities with a Complex Setup

This section contains the results of experiments measuring the achieved stream qualities using complex stream setups with the different workloads. The setups consist of mandatory streams and optional streams read with a bandwidth of 1 MByte/s and 2 MByte/s using a period length of 1000 ms and a request size of 64 KByte, and streams read with a bandwidth of 512 KByte/s and 384 KByte/s using a period length of 4000 ms and a request size of 16 KByte. The setups are created by adding optional streams with various qualities to the setup as long as those streams are accepted by the admission control.

A.1.1 IBM Ultrastar 36Z15



(a) 16 KByte Request Size.



(b) 64 KByte Request Size.

Request Size	Linear			Pattern-nf			Pattern-hf			Random		
	mean	dev	max	mean	dev	max	mean	dev	max	mean	dev	max
16 KByte	0.91	1.31	8.80	0.94	1.36	10.80	1.18	1.54	10.80	4.03	1.03	11.20
64 KByte	1.71	0.97	11.30	1.88	1.09	12.40	2.08	1.21	11.50	4.24	0.73	11.60

(c) Distribution Parameters Mean Value, Standard Deviation, and Maximum Value (in ms).

Figure A.1: Execution-Time Distributions, IBM Ultrastar 36Z15.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	24	64	1.00	726.02	1.00	0.0	1536.0
1	1000	32	64	0.98	56.40	1.00	2.0	2047.2
2	1000	32	64	0.98	56.40	1.00	2.0	2046.9
3	1000	32	64	0.98	56.40	1.00	2.0	2039.0
4	1000	16	64	0.98	29.20	0.98	0.0	1000.8
5	1000	16	64	0.98	29.20	0.98	0.0	999.9
6	1000	16	64	0.98	29.20	0.99	1.0	1010.1
7	1000	32	64	0.95	52.40	0.97	2.1	1990.6
8	1000	32	64	0.95	52.40	1.00	5.3	2044.8
9	1000	32	64	0.95	52.40	1.00	5.3	2038.6
10	1000	16	64	0.95	26.60	0.97	2.1	996.9
11	1000	16	64	0.95	26.60	0.92	-3.2	938.2
12	1000	16	64	0.95	26.60	0.98	3.2	998.6
13	1000	32	64	0.90	48.60	0.98	8.9	2001.5
14	1000	32	64	0.90	48.60	0.95	5.6	1951.8
15	1000	32	64	0.90	48.60	0.93	3.3	1898.0
16	1000	16	64	0.90	24.10	0.91	1.1	936.1
17	1000	16	64	0.90	24.10	0.87	-3.3	888.8
18	1000	16	64	0.90	24.10	0.85	-5.6	865.7
19	1000	32	64	0.80	42.60	0.78	-2.5	1592.5
20	1000	32	64	0.80	42.60	0.86	7.5	1761.3
21	1000	16	64	0.80	20.80	0.73	-8.8	744.3
22	1000	16	64	0.80	20.80	0.73	-8.8	747.7
23	4000	34	16	1.00	1028.53	1.00	0.0	136.0
24	4000	16	16	0.98	20.20	1.00	2.0	64.0
25	4000	16	16	0.98	20.20	1.00	2.0	63.8
26	4000	16	16	0.98	20.20	1.00	2.0	63.8
27	4000	12	16	0.98	16.20	0.99	1.0	47.3
28	4000	12	16	0.98	16.20	0.98	0.0	46.9
29	4000	12	16	0.98	16.20	0.98	0.0	47.2
30	4000	16	16	0.95	17.20	1.00	5.3	63.7
31	4000	16	16	0.95	17.20	0.99	4.2	63.3
32	4000	16	16	0.95	17.20	0.98	3.2	63.0
33	4000	12	16	0.95	13.50	0.96	1.1	46.3
34	4000	12	16	0.95	13.50	0.99	4.2	47.4
35	4000	12	16	0.95	13.50	0.97	2.1	46.6
36	4000	16	16	0.90	14.60	0.92	2.2	59.0
37	4000	16	16	0.90	14.60	0.95	5.6	60.9
38	4000	16	16	0.90	14.60	0.98	8.9	62.5
39	4000	12	16	0.90	11.40	0.83	-7.8	40.0
40	4000	12	16	0.90	11.40	0.89	-1.1	42.9
41	4000	12	16	0.90	11.40	0.92	2.2	44.0
42	4000	16	16	0.80	11.50	0.93	16.2	59.7
43	4000	16	16	0.80	11.50	0.88	10.0	56.2
44	4000	12	16	0.80	8.70	0.76	-5.0	36.6
45	4000	12	16	0.80	8.70	0.73	-8.8	35.0
						Average Deviation Δq_{dev}	3.9	
						Mean Value $\overline{q_{dev}}$	1.5	
						Standard Deviation σ	4.9	
						Best-Effort Requests (64 KByte Request Size)		745.2
						Best-Effort Requests (16 KByte Request Size)		224.6
						Total Bandwidth		35341.5
						Achieved Disk Utilization of Real-Time Streams		0.97

Table A.1: IBM Ultrastar 36Z15, Linear Workload.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	24	64	1.00	726.02	1.00	0.0	1536.0
1	1000	32	64	0.98	62.20	0.96	-2.0	1968.0
2	1000	32	64	0.98	62.20	0.99	1.0	2031.8
3	1000	32	64	0.98	62.20	0.98	0.0	2010.5
4	1000	16	64	0.98	32.20	0.99	1.0	1011.2
5	1000	16	64	0.98	32.20	0.94	-4.1	961.9
6	1000	16	64	0.98	32.20	0.99	1.0	1011.6
7	1000	32	64	0.95	57.80	0.96	1.1	1963.7
8	1000	32	64	0.95	57.80	0.96	1.1	1961.8
9	1000	32	64	0.95	57.80	0.99	4.2	2025.0
10	1000	16	64	0.95	29.40	0.97	2.1	991.1
11	1000	16	64	0.95	29.40	0.95	0.0	974.1
12	1000	16	64	0.95	29.40	0.96	1.1	982.4
13	1000	32	64	0.90	53.40	0.93	3.3	1903.8
14	1000	32	64	0.90	53.40	0.94	4.4	1920.6
15	1000	16	64	0.90	26.60	0.89	-1.1	913.1
16	1000	16	64	0.90	26.60	0.91	1.1	935.5
17	1000	32	64	0.80	47.00	0.79	-1.3	1627.1
18	1000	32	64	0.80	47.00	0.81	1.3	1660.4
19	1000	16	64	0.80	22.90	0.76	-5.0	781.4
20	4000	34	16	1.00	1028.53	1.00	0.0	136.0
21	4000	16	16	0.98	21.10	1.00	2.0	64.0
22	4000	16	16	0.98	21.10	1.00	2.0	63.7
23	4000	16	16	0.98	21.10	1.00	2.0	64.0
24	4000	12	16	0.98	16.80	0.99	1.0	47.5
25	4000	12	16	0.98	16.80	1.00	2.0	47.9
26	4000	12	16	0.98	16.80	0.97	-1.0	46.7
27	4000	16	16	0.95	18.00	0.96	1.1	61.6
28	4000	16	16	0.95	18.00	0.98	3.2	62.8
29	4000	16	16	0.95	18.00	0.99	4.2	63.2
30	4000	12	16	0.95	14.00	0.95	0.0	45.4
31	4000	12	16	0.95	14.00	0.91	-4.2	43.8
32	4000	12	16	0.95	14.00	0.97	2.1	46.5
33	4000	16	16	0.90	15.10	0.98	8.9	62.6
34	4000	16	16	0.90	15.10	0.98	8.9	62.9
35	4000	12	16	0.90	11.80	0.93	3.3	44.6
36	4000	12	16	0.90	11.80	0.88	-2.2	42.1
37	4000	16	16	0.80	11.90	0.84	5.0	53.9
38	4000	16	16	0.80	11.90	0.81	1.3	52.1
39	4000	12	16	0.80	9.00	0.70	-12.5	33.4
						Average Deviation Δq_{dev}	2.6	
						Mean Value $\overline{q_{dev}}$	0.9	
						Standard Deviation σ	3.6	
							Best-Effort Requests (64 KByte Request Size)	1183.2
							Best-Effort Requests (16 KByte Request Size)	785.7
							Total Bandwidth	32284.6
							Achieved Disk Utilization of Real-Time Streams	0.94

Table A.2: IBM Ultrastar 36Z15, Pattern-nf Workload.

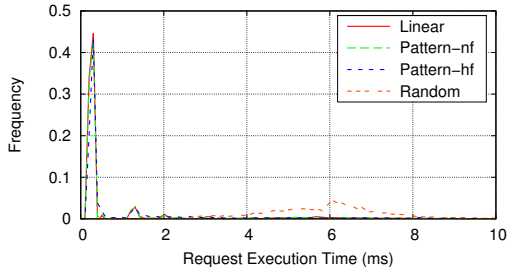
Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	24	64	1.00	726.02	1.00	0.0	1536.0
1	1000	32	64	0.98	68.80	0.95	-3.1	1950.9
2	1000	32	64	0.98	68.80	0.98	0.0	2009.8
3	1000	32	64	0.98	68.80	0.90	-8.2	1845.8
4	1000	16	64	0.98	35.50	1.00	2.0	1020.8
5	1000	16	64	0.98	35.50	0.99	1.0	1014.8
6	1000	16	64	0.98	35.50	0.97	-1.0	990.7
7	1000	32	64	0.95	64.00	0.85	-10.5	1734.8
8	1000	32	64	0.95	64.00	1.00	5.3	2040.5
9	1000	16	64	0.95	32.40	0.98	3.2	1000.1
10	1000	16	64	0.95	32.40	0.94	-1.1	961.1
11	1000	32	64	0.90	59.10	0.80	-11.1	1646.9
12	1000	32	64	0.90	59.10	0.99	10.0	2021.5
13	1000	16	64	0.90	29.40	0.86	-4.4	882.1
14	1000	16	64	0.90	29.40	0.85	-5.6	874.9
15	1000	32	64	0.80	52.00	0.72	-10.0	1470.9
16	1000	32	64	0.80	52.00	0.91	13.7	1870.9
17	1000	16	64	0.80	25.40	0.84	5.0	861.0
18	1000	16	64	0.80	25.40	0.79	-1.3	812.0
19	4000	34	16	1.00	1028.53	1.00	0.0	136.0
20	4000	16	16	0.98	25.40	1.00	2.0	63.7
21	4000	16	16	0.98	25.40	0.99	1.0	63.4
22	4000	16	16	0.98	25.40	1.00	2.0	64.0
23	4000	12	16	0.98	20.00	0.98	0.0	47.2
24	4000	12	16	0.98	20.00	1.00	2.0	48.0
25	4000	12	16	0.98	20.00	0.99	1.0	47.5
26	4000	16	16	0.95	21.70	0.99	4.2	63.3
27	4000	16	16	0.95	21.70	1.00	5.3	63.7
28	4000	12	16	0.95	17.00	0.98	3.2	46.9
29	4000	12	16	0.95	17.00	0.97	2.1	46.6
30	4000	16	16	0.90	18.60	0.86	-4.4	55.0
31	4000	16	16	0.90	18.60	1.00	11.1	63.7
32	4000	12	16	0.90	14.40	0.94	4.4	45.1
33	4000	12	16	0.90	14.40	0.82	-8.9	39.5
34	4000	16	16	0.80	14.90	0.92	15.0	58.8
35	4000	16	16	0.80	14.90	0.87	8.7	55.7
36	4000	12	16	0.80	11.20	0.93	16.2	44.9
37	4000	12	16	0.80	11.20	0.81	1.3	38.7
						Average Deviation Δq_{dev}	5.0	
						Mean Value $\overline{q_{dev}}$	1.3	
						Standard Deviation σ	6.6	
							Best-Effort Requests (64 KByte Request Size)	1168.4
							Best-Effort Requests (16 KByte Request Size)	616.7
							Total Bandwidth	29422.5
							Achieved Disk Utilization of Real-Time Streams	0.94

Table A.3: IBM Ultrastar 36Z15, Pattern-hf Workload.

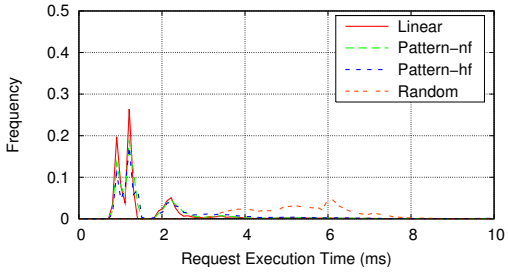
Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	24	64	1.00	726.02	1.00	0.0	1536.0
1	1000	32	64	0.98	131.70	0.99	1.0	2028.8
2	1000	16	64	0.98	65.20	0.99	1.0	1013.5
3	1000	32	64	0.95	127.00	0.97	2.1	1980.2
4	1000	16	64	0.95	62.60	0.96	1.1	987.7
5	1000	32	64	0.90	120.10	0.90	0.0	1853.0
6	1000	16	64	0.90	59.00	0.89	-1.1	907.1
7	1000	32	64	0.80	106.50	0.78	-2.5	1591.5
8	4000	34	16	1.00	1028.53	1.00	0.0	136.0
9	4000	16	16	0.98	63.00	0.94	-4.1	60.3
10	4000	12	16	0.98	47.10	0.98	0.0	47.1
11	4000	16	16	0.95	59.80	0.99	4.2	63.1
12	4000	12	16	0.95	44.60	0.95	0.0	45.8
13	4000	16	16	0.90	56.10	0.87	-3.3	55.5
14	4000	12	16	0.90	41.60	0.90	0.0	43.3
15	4000	16	16	0.80	49.60	0.86	7.5	54.9
						Average Deviation Δq_{dev}	1.7	
						Mean Value $\overline{q_{dev}}$	0.4	
						Standard Deviation σ	2.7	
						Best-Effort Requests (64 KByte Request Size)		645.3
						Best-Effort Requests (16 KByte Request Size)		148.0
						Total Bandwidth		13197.2
						Achieved Disk Utilization of Real-Time Streams		0.94

Table A.4: IBM Ultrastar 36Z15, Random Workload.

A.1.2 Seagate Cheetah 36ES



(a) 16 KByte Request Size.



(b) 64 KByte Request Size.

Request	Linear			Pattern-nf			Pattern-hf			Random		
Size	mean	dev	max	mean	dev	max	mean	dev	max	mean	dev	max
16 KByte	1.04	1.93	12.80	1.04	1.87	15.30	1.41	2.32	18.50	5.40	1.55	14.00
64 KByte	1.75	1.45	36.90	1.92	1.53	36.80	2.24	1.77	36.90	5.93	1.31	36.00

(c) Distribution Parameters Mean Value, Standard Deviation, and Maximum Value (in ms).

Figure A.2: Execution-Time Distributions, Seagate Cheetah 36ES.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	16	64	1.00	652.18	1.00	0.0	1024.0
1	1000	32	64	0.98	61.30	1.00	2.0	2046.9
2	1000	32	64	0.98	61.30	1.00	2.0	2046.7
3	1000	32	64	0.98	61.30	1.00	2.0	2044.4
4	1000	16	64	0.98	32.70	0.99	1.0	1013.8
5	1000	16	64	0.98	32.70	0.97	-1.0	998.0
6	1000	16	64	0.98	32.70	0.98	0.0	1008.4
7	1000	32	64	0.95	55.70	0.98	3.2	2014.7
8	1000	32	64	0.95	55.70	0.99	4.2	2029.9
9	1000	32	64	0.95	55.70	1.00	5.3	2044.8
10	1000	16	64	0.95	28.80	0.98	3.2	1004.2
11	1000	16	64	0.95	28.80	0.95	0.0	969.4
12	1000	16	64	0.95	28.80	0.96	1.1	985.2
13	1000	32	64	0.90	50.60	0.98	8.9	2001.1
14	1000	32	64	0.90	50.60	0.96	6.7	1969.1
15	1000	32	64	0.90	50.60	0.95	5.6	1949.4
16	1000	16	64	0.90	25.40	0.91	1.1	932.3
17	1000	16	64	0.90	25.40	0.90	0.0	925.2
18	1000	32	64	0.80	43.70	0.82	2.5	1676.6
19	1000	32	64	0.80	43.70	0.78	-2.5	1593.0
20	1000	16	64	0.80	21.40	0.78	-2.5	800.2
21	1000	16	64	0.80	21.40	0.73	-8.8	752.2
22	4000	16	16	1.00	652.18	1.00	0.0	64.0
23	4000	16	16	0.98	26.10	1.00	2.0	64.0
24	4000	16	16	0.98	26.10	1.00	2.0	64.0
25	4000	16	16	0.98	26.10	1.00	2.0	64.0
26	4000	12	16	0.98	21.10	0.99	1.0	47.7
27	4000	12	16	0.98	21.10	1.00	2.0	47.9
28	4000	12	16	0.98	21.10	0.99	1.0	47.7
29	4000	16	16	0.95	21.80	1.00	5.3	64.0
30	4000	16	16	0.95	21.80	0.98	3.2	63.0
31	4000	16	16	0.95	21.80	0.99	4.2	63.6
32	4000	12	16	0.95	17.20	0.92	-3.2	44.4
33	4000	12	16	0.95	17.20	0.95	0.0	45.7
34	4000	16	16	0.90	17.90	0.97	7.8	62.2
35	4000	16	16	0.90	17.90	0.96	6.7	61.5
36	4000	12	16	0.90	14.00	0.94	4.4	45.3
37	4000	12	16	0.90	14.00	0.93	3.3	44.4
38	4000	16	16	0.80	13.60	0.85	6.2	54.3
39	4000	16	16	0.80	13.60	0.89	11.2	56.9
40	4000	12	16	0.80	10.20	0.69	-13.8	33.2
41	4000	12	16	0.80	10.20	0.75	-6.3	36.1
						Average Deviation Δq_{dev}	3.6	
						Mean Value $\overline{q_{dev}}$	1.7	
						Standard Deviation σ	4.4	
							Best-Effort Requests (64 KByte Request Size)	1409.9
							Best-Effort Requests (16 KByte Request Size)	111.3
							Total Bandwidth	34424.5
							Achieved Disk Utilization of Real-Time Streams	0.96

Table A.5: Seagate Cheetah 36ES, Linear Workload.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	16	64	1.00	652.18	1.00	0.0	1024.0
1	1000	32	64	0.98	66.80	0.96	-2.0	1967.6
2	1000	32	64	0.98	66.80	0.99	1.0	2027.7
3	1000	32	64	0.98	66.80	0.98	0.0	2013.2
4	1000	16	64	0.98	35.40	0.98	0.0	1008.4
5	1000	16	64	0.98	35.40	0.91	-7.1	935.2
6	1000	16	64	0.98	35.40	0.98	0.0	1006.5
7	1000	32	64	0.95	60.70	0.96	1.1	1965.4
8	1000	32	64	0.95	60.70	0.96	1.1	1960.5
9	1000	32	64	0.95	60.70	0.99	4.2	2029.9
10	1000	16	64	0.95	31.30	0.94	-1.1	963.2
11	1000	16	64	0.95	31.30	0.94	-1.1	966.8
12	1000	32	64	0.90	55.30	0.95	5.6	1936.8
13	1000	32	64	0.90	55.30	0.91	1.1	1865.2
14	1000	16	64	0.90	27.80	0.87	-3.3	892.2
15	1000	16	64	0.90	27.80	0.83	-7.8	846.3
16	1000	32	64	0.80	48.00	0.87	8.7	1783.7
17	1000	32	64	0.80	48.00	0.91	13.7	1867.7
18	1000	16	64	0.80	24.80	0.77	-3.8	790.6
19	1000	16	64	0.80	24.80	0.69	-13.8	703.6
20	4000	16	16	1.00	652.18	1.00	0.0	64.0
21	4000	16	16	0.98	25.80	1.00	2.0	63.7
22	4000	16	16	0.98	25.80	1.00	2.0	64.0
23	4000	16	16	0.98	25.80	0.97	-1.0	62.2
24	4000	12	16	0.98	20.80	1.00	2.0	47.8
25	4000	12	16	0.98	20.80	1.00	2.0	48.0
26	4000	12	16	0.98	20.80	0.99	1.0	47.3
27	4000	16	16	0.95	21.40	0.93	-2.1	59.5
28	4000	16	16	0.95	21.40	0.98	3.2	62.8
29	4000	16	16	0.95	21.40	0.98	3.2	62.9
30	4000	12	16	0.95	17.00	0.90	-5.3	43.0
31	4000	12	16	0.95	17.00	0.94	-1.1	45.2
32	4000	16	16	0.90	17.70	0.90	0.0	57.4
33	4000	16	16	0.90	17.70	0.97	7.8	61.9
34	4000	12	16	0.90	13.80	0.86	-4.4	41.1
35	4000	12	16	0.90	13.80	0.92	2.2	44.4
36	4000	16	16	0.80	13.40	0.78	-2.5	49.9
37	4000	16	16	0.80	13.40	0.84	5.0	53.9
38	4000	12	16	0.80	11.30	0.83	3.7	40.0
39	4000	12	16	0.80	11.30	0.74	-7.5	35.6
						Average Deviation Δq_{dev}	3.4	
						Mean Value $\overline{q_{dev}}$	0.2	
						Standard Deviation σ	4.8	
						Best-Effort Requests (64 KByte Request Size)		718.5
						Best-Effort Requests (16 KByte Request Size)		822.4
							Total Bandwidth	31150.1
						Achieved Disk Utilization of Real-Time Streams		0.95

Table A.6: Seagate Cheetah 36ES, Pattern-nf Workload.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	16	64	1.00	652.18	1.00	0.0	1024.0
1	1000	32	64	0.98	77.60	0.96	-2.0	1965.7
2	1000	32	64	0.98	77.60	0.98	0.0	2012.6
3	1000	32	64	0.98	77.60	0.92	-6.1	1894.2
4	1000	16	64	0.98	41.00	1.00	2.0	1022.7
5	1000	16	64	0.98	41.00	0.99	1.0	1017.6
6	1000	32	64	0.95	70.80	0.97	2.1	1986.6
7	1000	32	64	0.95	70.80	0.89	-6.3	1831.7
8	1000	16	64	0.95	36.50	0.99	4.2	1012.9
9	1000	16	64	0.95	36.50	0.98	3.2	1004.8
10	1000	32	64	0.90	64.50	0.96	6.7	1972.3
11	1000	32	64	0.90	64.50	0.79	-12.2	1618.6
12	1000	16	64	0.90	32.40	0.96	6.7	979.8
13	1000	16	64	0.90	32.40	0.95	5.6	972.0
14	1000	32	64	0.80	56.00	0.85	6.2	1743.6
15	1000	32	64	0.80	56.00	0.70	-12.5	1426.1
16	1000	16	64	0.80	27.40	0.84	5.0	861.4
17	1000	16	64	0.80	28.20	0.69	-13.8	708.0
18	4000	16	16	1.00	652.18	1.00	0.0	64.0
19	4000	16	16	0.98	33.40	0.99	1.0	63.6
20	4000	16	16	0.98	33.40	1.00	2.0	64.0
21	4000	12	16	0.98	26.60	0.98	0.0	46.8
22	4000	12	16	0.98	26.60	1.00	2.0	48.0
23	4000	16	16	0.95	28.00	1.00	5.3	63.9
24	4000	16	16	0.95	28.00	0.98	3.2	62.5
25	4000	12	16	0.95	22.00	0.94	-1.1	45.2
26	4000	12	16	0.95	22.00	0.98	3.2	46.8
27	4000	16	16	0.90	23.40	0.99	10.0	63.4
28	4000	16	16	0.90	23.40	0.98	8.9	62.5
29	4000	12	16	0.90	18.20	0.82	-8.9	39.5
30	4000	12	16	0.90	18.30	0.97	7.8	46.4
31	4000	16	16	0.80	18.00	0.90	12.5	57.8
32	4000	16	16	0.80	18.00	0.89	11.2	56.8
33	4000	12	16	0.80	15.50	0.87	8.7	41.9
34	4000	12	16	0.80	15.50	0.82	2.5	39.4
						Average Deviation Δq_{dev}	5.3	
						Mean Value $\overline{q_{dev}}$	1.7	
						Standard Deviation σ	6.4	
							Best-Effort Requests (64 KByte Request Size)	734.7
							Best-Effort Requests (16 KByte Request Size)	472.8
							Total Bandwidth	27174.5
							Achieved Disk Utilization of Real-Time Streams	0.96

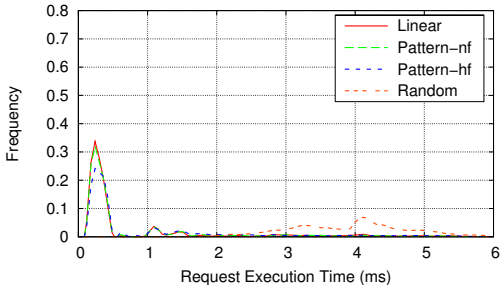
Table A.7: Seagate Cheetah 36ES, Pattern-hf Workload.

Data Stream (i)	Period Length (ms)	Number of Requests (a_i)	Request Size (KByte)	Requested Quality (q_i)	Reservation Time (ms)	Achieved Quality ($q_{ach,i}$)	Rel. Deviation of Quality ($q_{dev,i}$, %)	Measured Bandwidth KByte/s
0	1000	16	64	1.00	652.18	1.00	0.0	1024.0
1	1000	32	64	0.98	169.10	0.98	0.0	2011.1
2	1000	16	64	0.98	84.10	0.98	0.0	1008.6
3	1000	32	64	0.95	162.00	0.95	0.0	1951.4
4	1000	32	64	0.90	153.00	0.89	-1.1	1813.5
5	1000	32	64	0.80	135.50	0.76	-5.0	1555.2
6	4000	16	16	1.00	652.18	1.00	0.0	64.0
7	4000	16	16	0.98	92.80	0.96	-2.0	61.3
8	4000	12	16	0.98	69.40	0.99	1.0	47.6
9	4000	16	16	0.95	88.00	0.95	0.0	60.9
10	4000	16	16	0.90	82.50	0.93	3.3	59.7
11	4000	16	16	0.80	72.90	0.83	3.7	53.3
						Average Deviation Δq_{dev}	1.4	
						Mean Value $\overline{q_{dev}}$	0.0	
						Standard Deviation σ	2.2	
						Best-Effort Requests (64 KByte Request Size)		315.9
						Best-Effort Requests (16 KByte Request Size)		74.0
							Total Bandwidth	10100.6
						Achieved Disk Utilization of Real-Time Streams		0.96

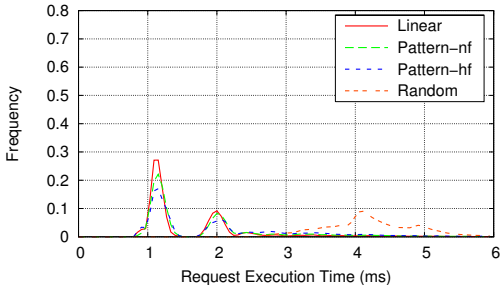
Table A.8: Seagate Cheetah 36ES, Random Workload.

A.2 Effect of the Distribution Class Width

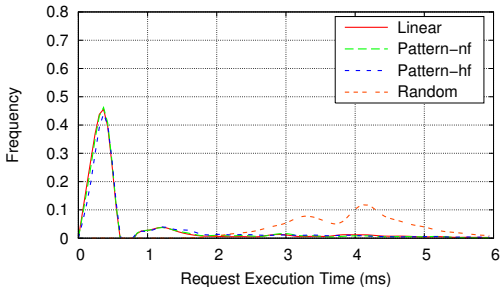
A.2.1 IBM Ultrastar 36Z15



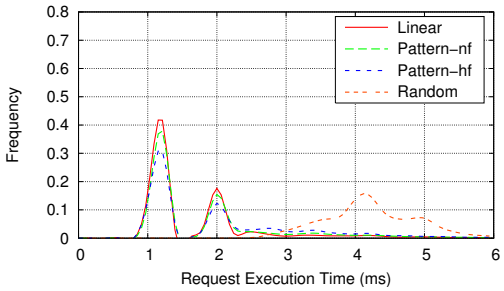
(a) 16 KByte Request Size, Class Width 100 ms.



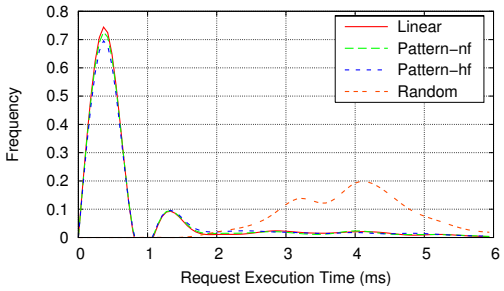
(b) 64 KByte Request Size, Class Width 100 ms.



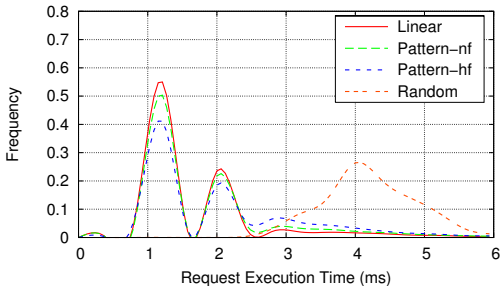
(c) 16 KByte Request Size, Class Width 200 ms.



(d) 64 KByte Request Size, Class Width 200 ms.



(e) 16 KByte Request Size, Class Width 400 ms.



(f) 64 KByte Request Size, Class Width 400 ms.

Req. Size / Class Width	Linear			Pattern-nf			Pattern-hf			Random		
	mean	dev	max	mean	dev	max	mean	dev	max	mean	dev	max
16 KByte:												
100 ms	0.91	1.30	8.80	0.94	1.36	10.80	1.19	1.54	10.80	4.03	1.03	11.20
200 ms	0.92	1.33	9.40	0.92	1.33	9.80	1.09	1.47	10.20	4.03	1.03	11.60
400 ms	0.91	1.30	9.20	0.95	1.35	10.80	1.05	1.45	9.60	4.04	1.04	10.80
64 KByte:												
100 ms	1.71	0.97	11.30	1.88	1.09	12.40	2.07	1.21	11.50	4.24	0.73	11.60
200 ms	1.72	0.97	10.60	1.86	1.08	11.80	2.07	1.21	11.80	4.24	0.73	12.00
400 ms	1.72	0.98	28.40	1.89	1.11	11.60	2.09	1.21	12.40	4.24	0.73	11.60

(g) Distribution Parameters Mean Value, Standard Deviation, and Maximum Value (in ms).

Figure A.3: Request Execution Time Distributions, IBM Ultrastar 36Z15.

				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	24	1.00	726.02	1.00	0.0	726.02	1.00	0.0	726.02	1.00	0.0
1	1000	32	0.98	56.40	1.00	2.0	57.20	1.00	2.0	57.60	1.00	2.0
2	1000	32	0.98	56.40	1.00	2.0	57.20	1.00	2.0	57.60	1.00	2.0
3	1000	32	0.98	56.40	1.00	2.0	57.20	1.00	2.0	57.60	1.00	2.0
4	1000	16	0.98	29.20	0.98	0.0	29.60	0.98	0.0	30.00	0.99	1.0
5	1000	16	0.98	29.20	0.98	0.0	29.60	0.98	0.0	30.00	0.99	1.0
6	1000	16	0.98	29.20	0.99	1.0	29.60	0.99	1.0	30.00	0.99	1.0
7	1000	32	0.95	52.40	0.97	2.1	53.20	0.97	2.1	53.60	0.99	4.2
8	1000	32	0.95	52.40	1.00	5.3	53.20	1.00	5.3	53.60	1.00	5.3
9	1000	32	0.95	52.40	1.00	5.3	53.20	1.00	5.3	53.60	1.00	5.3
10	1000	16	0.95	26.60	0.97	2.1	27.00	0.96	1.1	27.20	0.97	2.1
11	1000	16	0.95	26.60	0.92	-3.2	27.00	0.95	0.0	27.20	0.94	-1.1
12	1000	16	0.95	26.60	0.98	3.2	27.00	0.98	3.2	27.20	0.98	3.2
13	1000	32	0.90	48.60	0.98	8.9	49.20	0.98	8.9	49.60	0.99	10.0
14	1000	32	0.90	48.60	0.95	5.6	49.20	0.96	6.7	49.60	0.97	7.8
15	1000	32	0.90	48.60	0.93	3.3	49.20	0.93	3.3	49.60	0.93	3.3
16	1000	16	0.90	24.10	0.91	1.1	24.80	0.93	3.3	24.80	0.92	2.2
17	1000	16	0.90	24.10	0.87	-3.3	24.80	0.89	-1.1	24.80	0.90	0.0
18	1000	16	0.90	24.10	0.85	-5.6	24.80	0.86	-4.4	24.80	0.87	-3.3
19	1000	32	0.80	42.60	0.78	-2.5	43.60	0.79	-1.3	44.00	0.79	-1.3
20	1000	32	0.80	42.60	0.86	7.5	43.60	0.87	8.7	44.00	0.88	10.0
21	1000	16	0.80	20.80	0.73	-8.8	21.20	0.74	-7.5	21.60	0.74	-7.5
22	1000	16	0.80	20.80	0.73	-8.8	21.20	0.73	-8.8	21.60	0.72	-10.0
23	4000	34	1.00	1028.53	1.00	0.0	1028.53	1.00	0.0	1028.53	1.00	0.0
24	4000	16	0.98	20.20	1.00	2.0	20.80	1.00	2.0	21.60	1.00	2.0
25	4000	16	0.98	20.20	1.00	2.0	20.80	1.00	2.0	21.60	1.00	2.0
26	4000	16	0.98	20.20	1.00	2.0	20.80	1.00	2.0	21.60	1.00	2.0
27	4000	12	0.98	16.20	0.99	1.0	16.80	0.99	1.0	17.20	1.00	2.0
28	4000	12	0.98	16.20	0.98	0.0	16.80	0.99	1.0	17.20	0.99	1.0
29	4000	12	0.98	16.20	0.98	0.0	16.80	0.99	1.0	17.20	1.00	2.0
30	4000	16	0.95	17.20	1.00	5.3	18.00	0.99	4.2	18.40	1.00	5.3
31	4000	16	0.95	17.20	0.99	4.2	18.00	0.96	1.1	18.40	1.00	5.3
32	4000	16	0.95	17.20	0.98	3.2	18.00	1.00	5.3	18.40	1.00	5.3
33	4000	12	0.95	13.50	0.96	1.1	14.00	0.98	3.2	14.40	0.99	4.2
34	4000	12	0.95	13.50	0.99	4.2	14.00	0.96	1.1	14.40	0.99	4.2
35	4000	12	0.95	13.50	0.97	2.1	14.00	0.95	0.0	14.40	0.97	2.1
36	4000	16	0.90	14.60	0.92	2.2	15.20	0.97	7.8	16.00	0.95	5.6
37	4000	16	0.90	14.60	0.95	5.6	15.20	0.94	4.4	16.00	0.95	5.6
38	4000	16	0.90	14.60	0.98	8.9	15.20	0.96	6.7	16.00	0.97	7.8
39	4000	12	0.90	11.40	0.83	-7.8	11.60	0.86	-4.4	12.40	0.91	1.1
40	4000	12	0.90	11.40	0.89	-1.1	11.60	0.90	0.0	12.40	0.92	2.2
41	4000	12	0.90	11.40	0.92	2.2	11.60	0.92	2.2	12.40	0.95	5.6
42	4000	16	0.80	11.50	0.93	16.2	12.00	0.89	11.2	12.80	0.92	15.0
43	4000	16	0.80	11.50	0.88	10.0	12.00	0.80	0.0	12.80	0.86	7.5
44	4000	12	0.80	8.70	0.76	-5.0	9.00	0.72	-10.0	9.60	0.79	-1.3
45	4000	12	0.80	8.70	0.73	-8.8	9.00	0.72	-10.0	9.60	0.74	-7.5
	Average Deviation Δq_{dev}					3.9			3.5			4.0
	Mean Value $\overline{q_{dev}}$					1.5			1.4			2.6
	Standard Deviation σ					4.9			4.5			4.4

Table A.9: IBM Ultrastar 36Z15, Linear Workload.

				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	24	1.00	726.02	1.00	0.0	726.02	1.00	0.0	726.02	1.00	0.0
1	1000	32	0.98	62.20	0.96	-2.0	62.00	1.00	2.0	63.20	0.96	-2.0
2	1000	32	0.98	62.20	0.99	1.0	62.00	0.99	1.0	63.20	1.00	2.0
3	1000	32	0.98	62.20	0.98	0.0	62.00	0.98	0.0	63.20	0.98	0.0
4	1000	16	0.98	32.20	0.99	1.0	32.00	0.98	0.0	32.80	0.97	-1.0
5	1000	16	0.98	32.20	0.94	-4.1	32.00	0.97	-1.0	32.80	0.92	-6.1
6	1000	16	0.98	32.20	0.99	1.0	32.00	0.98	0.0	32.80	0.99	1.0
7	1000	32	0.95	57.80	0.96	1.1	57.60	0.96	1.1	58.80	0.96	1.1
8	1000	32	0.95	57.80	0.96	1.1	57.60	0.96	1.1	58.80	0.98	3.2
9	1000	32	0.95	57.80	0.99	4.2	57.60	0.94	-1.1	58.80	0.95	0.0
10	1000	16	0.95	29.40	0.97	2.1	29.20	0.95	0.0	30.00	0.96	1.1
11	1000	16	0.95	29.40	0.95	0.0	29.20	0.95	0.0	30.00	0.98	3.2
12	1000	16	0.95	29.40	0.96	1.1	29.20	0.99	4.2	30.00	0.98	3.2
13	1000	32	0.90	53.40	0.93	3.3	53.20	0.97	7.8	54.40	0.95	5.6
14	1000	32	0.90	53.40	0.94	4.4	53.20	0.92	2.2	54.40	0.94	4.4
15	1000	16	0.90	26.60	0.89	-1.1	26.60	0.85	-5.6	27.20	0.88	-2.2
16	1000	16	0.90	26.60	0.91	1.1	26.60	0.92	2.2	27.20	0.94	4.4
17	1000	32	0.80	47.00	0.79	-1.3	46.80	0.87	8.7	47.60	0.85	6.2
18	1000	32	0.80	47.00	0.81	1.3	46.80	0.82	2.5	47.60	0.81	1.3
19	1000	16	0.80	22.90	0.76	-5.0	23.00	0.76	-5.0	24.00	0.81	1.3
20	4000	34	1.00	1028.53	1.00	0.0	1028.53	1.00	0.0	1028.53	1.00	0.0
21	4000	16	0.98	21.10	1.00	2.0	21.00	1.00	2.0	22.00	1.00	2.0
22	4000	16	0.98	21.10	1.00	2.0	21.00	1.00	2.0	22.00	1.00	2.0
23	4000	16	0.98	21.10	1.00	2.0	21.00	1.00	2.0	22.00	1.00	2.0
24	4000	12	0.98	16.80	0.99	1.0	16.80	0.98	0.0	17.60	0.99	1.0
25	4000	12	0.98	16.80	1.00	2.0	16.80	1.00	2.0	17.60	0.99	1.0
26	4000	12	0.98	16.80	0.97	-1.0	16.80	0.99	1.0	17.60	0.98	0.0
27	4000	16	0.95	18.00	0.96	1.1	18.00	0.97	2.1	19.20	1.00	5.3
28	4000	16	0.95	18.00	0.98	3.2	18.00	0.99	4.2	19.20	0.95	0.0
29	4000	16	0.95	18.00	0.99	4.2	18.00	0.99	4.2	19.20	0.99	4.2
30	4000	12	0.95	14.00	0.95	0.0	14.00	0.96	1.1	14.80	0.99	4.2
31	4000	12	0.95	14.00	0.91	-4.2	14.00	0.98	3.2	14.80	0.97	2.1
32	4000	12	0.95	14.00	0.97	2.1	14.00	0.96	1.1	14.80	0.98	3.2
33	4000	16	0.90	15.10	0.98	8.9	15.20	0.98	8.9	16.40	0.96	6.7
34	4000	16	0.90	15.10	0.98	8.9	15.20	0.98	8.9	16.40	0.98	8.9
35	4000	12	0.90	11.80	0.93	3.3	11.60	0.91	1.1	12.80	0.95	5.6
36	4000	12	0.90	11.80	0.88	-2.2	11.60	0.86	-4.4	12.80	0.95	5.6
37	4000	16	0.80	11.90	0.84	5.0	12.00	0.84	5.0	13.20	0.92	15.0
38	4000	16	0.80	11.90	0.81	1.3	12.00	0.81	1.3	13.20	0.82	2.5
39	4000	12	0.80	9.00	0.70	-12.5	9.00	0.71	-11.3	10.40	0.78	-2.5
	Average Deviation Δq_{dev}					2.6			2.8			3.1
	Mean Value $\overline{q_{dev}}$					0.9			1.4			2.4
	Standard Deviation σ					3.6			3.8			3.5

Table A.10: IBM Ultrastar 36Z15, Pattern-nf Workload.

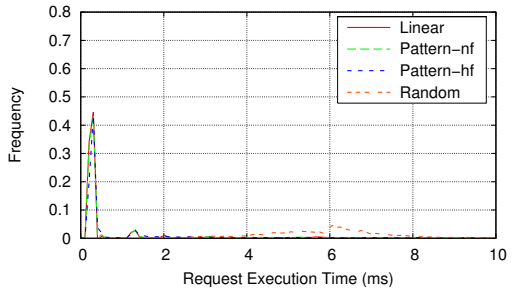
				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	24	1.00	726.02	1.00	0.0	726.02	1.00	0.0	726.02	1.00	0.0
1	1000	32	0.98	68.80	0.95	-3.1	68.80	0.99	1.0	69.20	0.97	-1.0
2	1000	32	0.98	68.80	0.98	0.0	68.80	0.97	-1.0	69.20	0.97	-1.0
3	1000	32	0.98	68.80	0.90	-8.2	68.80	0.89	-9.2	69.20	0.89	-9.2
4	1000	16	0.98	35.50	1.00	2.0	35.60	0.99	1.0	36.00	0.99	1.0
5	1000	16	0.98	35.50	0.99	1.0	35.60	0.97	-1.0	36.00	0.99	1.0
6	1000	16	0.98	35.50	0.97	-1.0	35.60	0.98	0.0	36.00	0.97	-1.0
7	1000	32	0.95	64.00	0.85	-10.5	64.00	0.86	-9.5	64.80	0.86	-9.5
8	1000	32	0.95	64.00	1.00	5.3	64.00	0.99	4.2	64.80	1.00	5.3
9	1000	16	0.95	32.40	0.98	3.2	32.60	0.99	4.2	32.80	0.90	-5.3
10	1000	16	0.95	32.40	0.94	-1.1	32.60	0.94	-1.1	32.80	0.94	-1.1
11	1000	32	0.90	59.10	0.80	-11.1	59.20	0.82	-8.9	60.00	0.81	-10.0
12	1000	32	0.90	59.10	0.99	10.0	59.20	0.99	10.0	60.00	1.00	11.1
13	1000	16	0.90	29.40	0.86	-4.4	29.60	0.94	4.4	30.00	0.89	-1.1
14	1000	16	0.90	29.40	0.85	-5.6	29.60	0.91	1.1	30.00	0.91	1.1
15	1000	32	0.80	52.00	0.72	-10.0	52.00	0.71	-11.3	52.40	0.73	-8.8
16	1000	32	0.80	52.00	0.91	13.7	52.00	0.94	17.5	52.40	0.93	16.2
17	1000	16	0.80	25.40	0.84	5.0	25.60	0.76	-5.0	26.00	0.87	8.7
18	1000	16	0.80	25.40	0.79	-1.3	25.60	0.76	-5.0	26.00	0.76	-5.0
19	4000	34	1.00	1028.53	1.00	0.0	1028.53	1.00	0.0	1028.53	1.00	0.0
20	4000	16	0.98	25.40	1.00	2.0	24.00	1.00	2.0	24.00	0.99	1.0
21	4000	16	0.98	25.40	0.99	1.0	24.00	1.00	2.0	24.00	1.00	2.0
22	4000	16	0.98	25.40	1.00	2.0	24.00	0.96	-2.0	24.00	0.98	0.0
23	4000	12	0.98	20.00	0.98	0.0	19.00	1.00	2.0	19.20	0.99	1.0
24	4000	12	0.98	20.00	1.00	2.0	19.00	1.00	2.0	19.20	0.99	1.0
25	4000	12	0.98	20.00	0.99	1.0	19.00	0.99	1.0	19.20	0.99	1.0
26	4000	16	0.95	21.70	0.99	4.2	20.80	0.95	0.0	20.80	0.97	2.1
27	4000	16	0.95	21.70	1.00	5.3	20.80	0.98	3.2	20.80	1.00	5.3
28	4000	12	0.95	17.00	0.98	3.2	16.00	0.95	0.0	16.00	0.98	3.2
29	4000	12	0.95	17.00	0.97	2.1	16.00	0.97	2.1	16.00	0.93	-2.1
30	4000	16	0.90	18.60	0.86	-4.4	17.60	0.97	7.8	17.60	0.90	0.0
31	4000	16	0.90	18.60	1.00	11.1	17.60	0.99	10.0	17.60	0.98	8.9
32	4000	12	0.90	14.40	0.94	4.4	13.60	0.94	4.4	13.60	0.80	-11.1
33	4000	12	0.90	14.40	0.82	-8.9	13.60	0.94	4.4	13.60	0.91	1.1
34	4000	16	0.80	14.90	0.92	15.0	14.00	0.88	10.0	14.40	0.71	-11.3
35	4000	16	0.80	14.90	0.87	8.7	14.00	0.84	5.0	14.40	0.93	16.2
36	4000	12	0.80	11.20	0.93	16.2	10.60	0.75	-6.3	10.80	0.71	-11.3
37	4000	12	0.80	11.20	0.81	1.3	10.60	0.74	-7.5	10.80	0.81	1.3
	Average Deviation Δq_{dev}					5.0			4.4			4.7
	Mean Value $\overline{q_{dev}}$					1.3			0.8			0.0
	Standard Deviation σ					6.6			5.9			6.7

Table A.11: IBM Ultrastar 36Z15, Pattern-hf Workload.

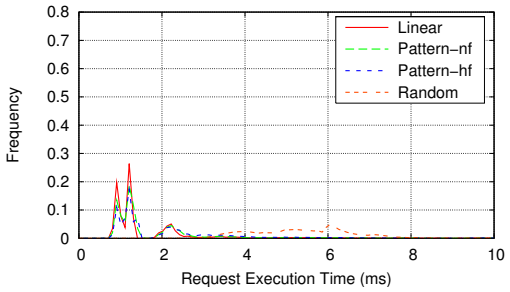
				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	24	1.00	726.02	1.00	0.0	726.02	1.00	0.0	726.02	1.00	0.0
1	1000	32	0.98	131.70	0.99	1.0	131.80	0.99	1.0	132.00	0.99	1.0
2	1000	16	0.98	65.20	0.99	1.0	65.40	0.99	1.0	65.20	0.99	1.0
3	1000	32	0.95	127.00	0.97	2.1	127.20	0.97	2.1	127.20	0.97	2.1
4	1000	16	0.95	62.60	0.96	1.1	62.80	0.96	1.1	63.20	0.97	2.1
5	1000	32	0.90	120.10	0.90	0.0	120.20	0.91	1.1	120.00	0.91	1.1
6	1000	16	0.90	59.00	0.89	-1.1	59.20	0.89	-1.1	59.20	0.89	-1.1
7	1000	32	0.80	106.50	0.78	-2.5	106.60	0.78	-2.5	106.40	0.78	-2.5
8	4000	34	1.00	1028.53	1.00	0.0	1028.53	1.00	0.0	1028.53	1.00	0.0
9	4000	16	0.98	63.00	0.94	-4.1	63.20	0.94	-4.1	63.60	0.95	-3.1
10	4000	12	0.98	47.10	0.98	0.0	47.20	0.98	0.0	47.60	0.98	0.0
11	4000	16	0.95	59.80	0.99	4.2	60.00	0.99	4.2	60.00	0.97	2.1
12	4000	12	0.95	44.60	0.95	0.0	44.80	0.96	1.1	44.80	0.94	-1.1
13	4000	16	0.90	56.10	0.87	-3.3	56.20	0.87	-3.3	56.40	0.87	-3.3
14	4000	12	0.90	41.60	0.90	0.0	42.00	0.91	1.1	42.00	0.92	2.2
15	4000	16	0.80	49.60	0.86	7.5	50.00	0.85	6.2	50.00	0.86	7.5
	Average Deviation Δq_{dev}					1.7			1.9			1.9
	Mean Value $\overline{q_{dev}}$					0.4			0.5			0.5
	Standard Deviation σ					2.7			2.5			2.5

Table A.12: IBM Ultrastar 36Z15, Random Workload.

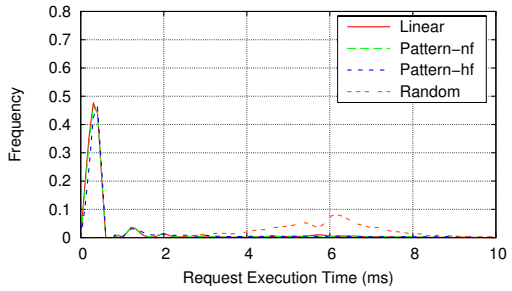
A.2.2 Seagate Cheetah 36ES



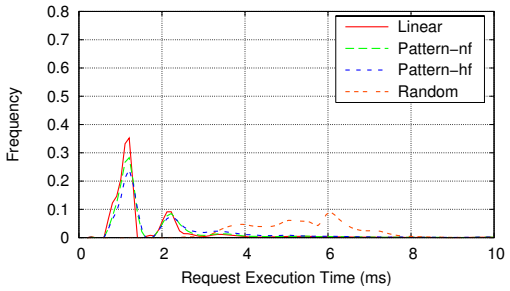
(a) 16 KByte Request Size, Class Width 100 ms.



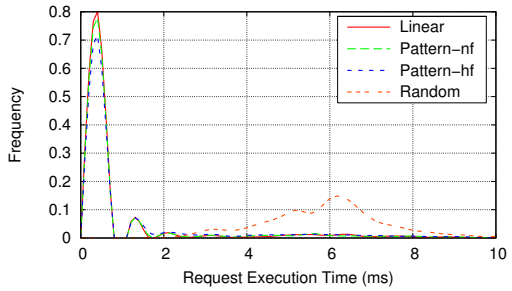
(b) 64 KByte Request Size, Class Width 100 ms.



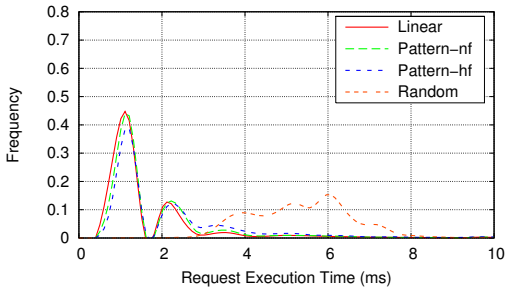
(c) 16 KByte Request Size, Class Width 200 ms.



(d) 64 KByte Request Size, Class Width 200 ms.



(e) 16 KByte Request Size, Class Width 400 ms.



(f) 64 KByte Request Size, Class Width 400 ms.

Req. Size / Class Width	Linear			Pattern-nf			Pattern-hf			Random		
	mean	dev	max	mean	dev	max	mean	dev	max	mean	dev	max
16 KByte:												
100 ms	1.04	1.93	12.80	1.04	1.87	15.30	1.41	2.32	18.50	5.93	1.55	14.00
200 ms	1.07	1.86	14.60	1.06	1.92	13.60	1.33	2.20	13.60	5.93	1.56	14.80
400 ms	1.01	1.88	12.80	1.09	1.97	13.60	1.31	2.13	13.20	5.95	1.57	14.00
64 KByte:												
100 ms	1.75	1.45	36.90	1.92	1.53	36.80	2.24	1.77	36.90	5.40	1.31	36.00
200 ms	1.76	1.45	36.80	1.93	1.56	36.80	2.24	1.76	37.00	5.38	1.29	35.00
400 ms	1.75	1.45	36.80	1.92	1.54	36.80	2.25	1.77	37.20	5.39	1.29	36.00

(g) Distribution Parameters Mean Value, Standard Deviation, and Maximum Value (in ms).

Figure A.4: Request Execution Time Distributions, Seagate Cheetah 36ES.

				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
1	1000	32	0.98	61.30	1.00	2.0	61.60	1.00	2.0	60.80	1.00	2.0
2	1000	32	0.98	61.30	1.00	2.0	61.60	1.00	2.0	60.80	1.00	2.0
3	1000	32	0.98	61.30	1.00	2.0	61.60	1.00	2.0	60.80	0.99	1.0
4	1000	16	0.98	32.70	0.99	1.0	32.80	0.99	1.0	32.80	1.00	2.0
5	1000	16	0.98	32.70	0.97	-1.0	32.80	0.99	1.0	32.80	0.99	1.0
6	1000	16	0.98	32.70	0.98	0.0	32.80	1.00	2.0	32.80	0.99	1.0
7	1000	32	0.95	55.70	0.98	3.2	56.00	0.98	3.2	55.20	0.98	3.2
8	1000	32	0.95	55.70	0.99	4.2	56.00	0.99	4.2	55.20	0.99	4.2
9	1000	32	0.95	55.70	1.00	5.3	56.00	1.00	5.3	55.20	0.99	4.2
10	1000	16	0.95	28.80	0.98	3.2	29.00	0.98	3.2	28.80	0.96	1.1
11	1000	16	0.95	28.80	0.95	0.0	29.00	0.94	-1.1	28.80	0.97	2.1
12	1000	16	0.95	28.80	0.96	1.1	29.00	0.96	1.1	28.80	0.96	1.1
13	1000	32	0.90	50.60	0.98	8.9	50.80	0.97	7.8	50.00	0.97	7.8
14	1000	32	0.90	50.60	0.96	6.7	50.80	0.96	6.7	50.00	0.95	5.6
15	1000	32	0.90	50.60	0.95	5.6	50.80	0.95	5.6	50.00	0.92	2.2
16	1000	16	0.90	25.40	0.91	1.1	25.60	0.92	2.2	25.20	0.91	1.1
17	1000	16	0.90	25.40	0.90	0.0	25.60	0.91	1.1	25.20	0.89	-1.1
18	1000	32	0.80	43.70	0.82	2.5	44.00	0.84	5.0	43.20	0.82	2.5
19	1000	32	0.80	43.70	0.78	-2.5	44.00	0.78	-2.5	43.20	0.80	0.0
20	1000	16	0.80	21.40	0.78	-2.5	21.60	0.79	-1.3	21.20	0.81	1.3
21	1000	16	0.80	21.40	0.73	-8.8	21.60	0.76	-5.0	21.20	0.74	-7.5
22	4000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
23	4000	16	0.98	26.10	1.00	2.0	25.80	1.00	2.0	26.80	1.00	2.0
24	4000	16	0.98	26.10	1.00	2.0	25.80	1.00	2.0	26.80	1.00	2.0
25	4000	16	0.98	26.10	1.00	2.0	25.80	1.00	2.0	26.80	1.00	2.0
26	4000	12	0.98	21.10	0.99	1.0	20.80	1.00	2.0	21.60	1.00	2.0
27	4000	12	0.98	21.10	1.00	2.0	20.80	1.00	2.0	21.60	1.00	2.0
28	4000	12	0.98	21.10	0.99	1.0	20.80	1.00	2.0	21.60	1.00	2.0
29	4000	16	0.95	21.80	1.00	5.3	21.60	1.00	5.3	22.40	1.00	5.3
30	4000	16	0.95	21.80	0.98	3.2	21.60	0.99	4.2	22.40	1.00	5.3
31	4000	16	0.95	21.80	0.99	4.2	21.60	0.97	2.1	22.40	0.98	3.2
32	4000	12	0.95	17.20	0.92	-3.2	17.20	0.93	-2.1	18.00	0.94	-1.1
33	4000	12	0.95	17.20	0.95	0.0	17.20	0.99	4.2	18.00	0.99	4.2
34	4000	16	0.90	17.90	0.97	7.8	18.00	0.97	7.8	18.80	0.97	7.8
35	4000	16	0.90	17.90	0.96	6.7	18.00	0.97	7.8	18.80	0.99	10.0
36	4000	12	0.90	14.00	0.94	4.4	14.00	0.92	2.2	14.80	0.97	7.8
37	4000	12	0.90	14.00	0.93	3.3	14.00	0.92	2.2	14.80	0.93	3.3
38	4000	16	0.80	13.60	0.85	6.2	13.60	0.78	-2.5	14.40	0.90	12.5
39	4000	16	0.80	13.60	0.89	11.2	13.60	0.87	8.7	14.40	0.91	13.7
40	4000	12	0.80	10.20	0.69	-13.8	10.20	0.69	-13.8	11.20	0.82	2.5
41	4000	12	0.80	10.20	0.75	-6.3	10.20	0.70	-12.5	11.20	0.78	-2.5
	Average Deviation Δq_{dev}					3.6			3.6			3.5
	Mean Value $\overline{q_{dev}}$					1.7			1.7			2.9
	Standard Deviation σ					4.4			4.4			3.7

Table A.13: Seagate Cheetah 36ES, Linear Workload.

				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
1	1000	32	0.98	66.80	0.96	-2.0	67.40	0.96	-2.0	66.40	0.99	1.0
2	1000	32	0.98	66.80	0.99	1.0	67.40	0.99	1.0	66.40	0.99	1.0
3	1000	32	0.98	66.80	0.98	0.0	67.40	0.98	0.0	66.40	0.98	0.0
4	1000	16	0.98	35.40	0.98	0.0	36.00	0.98	0.0	36.00	1.00	2.0
5	1000	16	0.98	35.40	0.91	-7.1	36.00	0.96	-2.0	36.00	0.99	1.0
6	1000	16	0.98	35.40	0.98	0.0	36.00	0.97	-1.0	36.00	0.99	1.0
7	1000	32	0.95	60.70	0.96	1.1	61.20	0.96	1.1	60.40	0.97	2.1
8	1000	32	0.95	60.70	0.96	1.1	61.20	0.96	1.1	60.40	0.97	2.1
9	1000	32	0.95	60.70	0.99	4.2	61.20	0.94	-1.1	60.40	0.94	-1.1
10	1000	16	0.95	31.30	0.94	-1.1	32.00	0.95	0.0	31.20	0.94	-1.1
11	1000	16	0.95	31.30	0.94	-1.1	32.00	0.93	-2.1	31.20	0.96	1.1
12	1000	32	0.90	55.30	0.95	5.6	56.00	0.94	4.4	55.20	0.94	4.4
13	1000	32	0.90	55.30	0.91	1.1	56.00	0.90	0.0	55.20	0.90	0.0
14	1000	16	0.90	27.80	0.87	-3.3	28.00	0.85	-5.6	28.00	0.87	-3.3
15	1000	16	0.90	27.80	0.83	-7.8	28.00	0.85	-5.6	28.00	0.84	-6.7
16	1000	32	0.80	48.00	0.87	8.7	48.20	0.89	11.2	48.00	0.90	12.5
17	1000	32	0.80	48.00	0.91	13.7	48.20	0.80	0.0	48.00	0.82	2.5
18	1000	16	0.80	24.80	0.77	-3.8	24.80	0.73	-8.8	24.80	0.78	-2.5
19	1000	16	0.80	24.80	0.69	-13.8	24.80	0.75	-6.3	24.80	0.79	-1.3
20	4000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
21	4000	16	0.98	25.80	1.00	2.0	26.80	1.00	2.0	28.40	1.00	2.0
22	4000	16	0.98	25.80	1.00	2.0	26.80	1.00	2.0	28.40	1.00	2.0
23	4000	16	0.98	25.80	0.97	-1.0	26.80	0.99	1.0	28.40	1.00	2.0
24	4000	12	0.98	20.80	1.00	2.0	21.60	0.99	1.0	22.80	1.00	2.0
25	4000	12	0.98	20.80	1.00	2.0	21.60	1.00	2.0	22.80	0.99	1.0
26	4000	12	0.98	20.80	0.99	1.0	21.60	1.00	2.0	22.80	1.00	2.0
27	4000	16	0.95	21.40	0.93	-2.1	22.40	0.95	0.0	24.00	0.97	2.1
28	4000	16	0.95	21.40	0.98	3.2	22.40	1.00	5.3	24.00	0.97	2.1
29	4000	16	0.95	21.40	0.98	3.2	22.40	0.99	4.2	24.00	1.00	5.3
30	4000	12	0.95	17.00	0.90	-5.3	18.00	0.97	2.1	18.80	0.98	3.2
31	4000	12	0.95	17.00	0.94	-1.1	18.00	0.98	3.2	18.80	0.97	2.1
32	4000	16	0.90	17.70	0.90	0.0	18.80	0.86	-4.4	20.00	0.97	7.8
33	4000	16	0.90	17.70	0.97	7.8	18.80	0.92	2.2	20.00	0.99	10.0
34	4000	12	0.90	13.80	0.86	-4.4	14.60	0.89	-1.1	16.00	0.97	7.8
35	4000	12	0.90	13.80	0.92	2.2	14.60	0.88	-2.2	16.00	0.98	8.9
36	4000	16	0.80	13.40	0.78	-2.5	14.20	0.91	13.7	16.00	0.93	16.2
37	4000	16	0.80	13.40	0.84	5.0	14.20	0.90	12.5	16.00	0.89	11.2
38	4000	12	0.80	11.30	0.83	3.7	11.40	0.81	1.3	12.00	0.65	-18.8
39	4000	12	0.80	11.30	0.74	-7.5	11.40	0.82	2.5	12.00	0.72	-10.0
	Average Deviation Δq_{dev}					3.4			3.0			4.1
	Mean Value $\overline{q_{dev}}$					0.2			0.8			1.9
	Standard Deviation σ					4.8			4.4			5.8

Table A.14: Seagate Cheetah 36ES, Pattern-nf Workload.

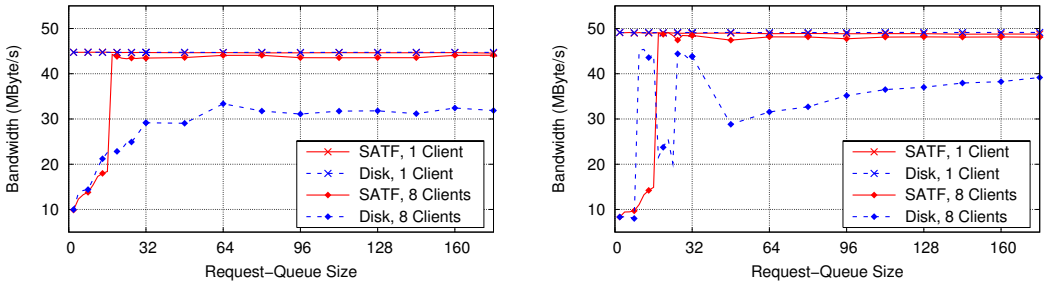
				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
1	1000	32	0.98	77.60	0.96	-2.0	77.60	0.99	1.0	77.60	0.98	0.0
2	1000	32	0.98	77.60	0.98	0.0	77.60	0.98	0.0	77.60	0.99	1.0
3	1000	32	0.98	77.60	0.92	-6.1	77.60	0.92	-6.1	77.60	0.90	-8.2
4	1000	16	0.98	41.00	1.00	2.0	41.00	1.00	2.0	41.60	1.00	2.0
5	1000	16	0.98	41.00	0.99	1.0	41.00	0.99	1.0	41.60	0.99	1.0
6	1000	32	0.95	70.80	0.97	2.1	70.80	0.98	3.2	71.20	0.96	1.1
7	1000	32	0.95	70.80	0.89	-6.3	70.80	0.86	-9.5	71.20	0.88	-7.4
8	1000	16	0.95	36.50	0.99	4.2	36.60	0.99	4.2	36.80	0.99	4.2
9	1000	16	0.95	36.50	0.98	3.2	36.60	0.88	-7.4	36.80	0.91	-4.2
10	1000	32	0.90	64.50	0.96	6.7	64.80	0.95	5.6	64.80	0.96	6.7
11	1000	32	0.90	64.50	0.79	-12.2	64.80	0.80	-11.1	64.80	0.82	-8.9
12	1000	16	0.90	32.40	0.96	6.7	32.40	0.96	6.7	32.80	0.97	7.8
13	1000	16	0.90	32.40	0.95	5.6	32.40	0.95	5.6	32.80	0.94	4.4
14	1000	32	0.80	56.00	0.85	6.2	56.00	0.88	10.0	56.00	0.90	12.5
15	1000	32	0.80	56.00	0.70	-12.5	56.00	0.71	-11.3	56.00	0.69	-13.8
16	1000	16	0.80	27.40	0.84	5.0	27.60	0.88	10.0	28.00	0.89	11.2
17	1000	16	0.80	28.20	0.69	-13.8	28.20	0.80	0.0	28.40	0.85	6.2
18	4000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
19	4000	16	0.98	33.40	0.99	1.0	32.00	0.99	1.0	32.00	1.00	2.0
20	4000	16	0.98	33.40	1.00	2.0	32.00	0.99	1.0	32.00	0.98	0.0
21	4000	12	0.98	26.60	0.98	0.0	25.60	0.98	0.0	25.60	0.97	-1.0
22	4000	12	0.98	26.60	1.00	2.0	25.60	1.00	2.0	25.60	0.99	1.0
23	4000	16	0.95	28.00	1.00	5.3	27.00	1.00	5.3	26.80	0.94	-1.1
24	4000	16	0.95	28.00	0.98	3.2	27.00	0.99	4.2	26.80	0.96	1.1
25	4000	12	0.95	22.00	0.94	-1.1	21.20	0.93	-2.1	21.20	0.94	-1.1
26	4000	12	0.95	22.00	0.98	3.2	21.20	0.95	0.0	21.20	0.99	4.2
27	4000	16	0.90	23.40	0.99	10.0	22.80	0.96	6.7	22.80	0.97	7.8
28	4000	16	0.90	23.40	0.98	8.9	22.80	0.97	7.8	22.80	0.98	8.9
29	4000	12	0.90	18.20	0.82	-8.9	17.60	0.80	-11.1	17.60	0.90	0.0
30	4000	12	0.90	18.30	0.97	7.8	17.60	0.96	6.7	17.60	0.94	4.4
31	4000	16	0.80	18.00	0.90	12.5	18.00	0.87	8.7	18.00	0.91	13.7
32	4000	16	0.80	18.00	0.89	11.2	18.00	0.84	5.0	18.00	0.86	7.5
33	4000	12	0.80	15.50	0.87	8.7	14.60	0.81	1.3	15.20	0.79	-1.3
34	4000	12	0.80	15.50	0.82	2.5	14.60	0.85	6.2	15.20	0.87	8.7
	Average Deviation Δq_{dev}					5.3			4.7			4.7
	Mean Value $\overline{q_{dev}}$					1.7			1.3			2.0
	Standard Deviation σ					6.4			5.8			6.0

Table A.15: Seagate Cheetah 36ES, Pattern-hf Workload.

				Class Width 0.10 ms			Class Width 0.20 ms			Class Width 0.40 ms		
	Period	#Req.	q	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}	r	q_{ach}	q_{dev}
0	1000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
1	1000	32	0.98	169.10	0.98	0.0	168.80	0.98	0.0	169.20	0.98	0.0
2	1000	16	0.98	84.10	0.98	0.0	84.00	0.98	0.0	84.00	0.98	0.0
3	1000	32	0.95	162.00	0.95	0.0	161.60	0.95	0.0	162.40	0.96	1.1
4	1000	32	0.90	153.00	0.89	-1.1	152.60	0.89	-1.1	152.80	0.88	-2.2
5	1000	32	0.80	135.50	0.76	-5.0	135.20	0.76	-5.0	136.00	0.76	-5.0
6	4000	16	1.00	652.18	1.00	0.0	652.18	1.00	0.0	652.18	1.00	0.0
7	4000	16	0.98	92.80	0.96	-2.0	92.80	0.97	-1.0	93.60	0.97	-1.0
8	4000	12	0.98	69.40	0.99	1.0	69.40	0.99	1.0	70.40	0.99	1.0
9	4000	16	0.95	88.00	0.95	0.0	88.00	0.96	1.1	88.40	0.96	1.1
10	4000	16	0.90	82.50	0.93	3.3	82.60	0.93	3.3	83.20	0.94	4.4
11	4000	16	0.80	72.90	0.83	3.7	73.00	0.84	5.0	73.20	0.83	3.7
	Average Deviation Δq_{dev}					1.4			1.5			1.6
	Mean Value $\overline{q_{dev}}$					-0.0			0.3			0.3
	Standard Deviation σ					2.2			2.3			2.4

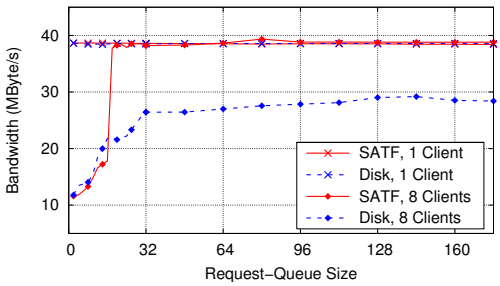
Table A.16: Seagate Cheetah 36ES, Random Workload.

A.3 Comparison of the SATF Scheduler and the Disk Scheduler

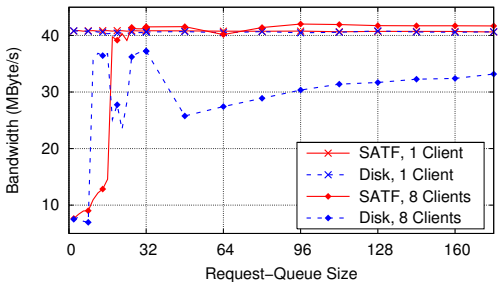


(a) IBM Ultrastar 36Z15, Linear Workload.

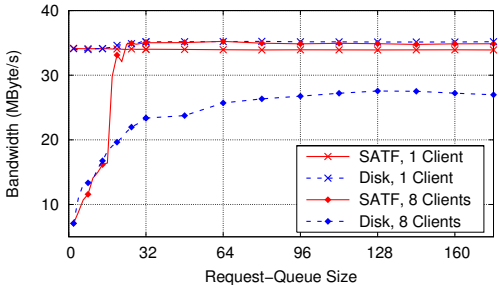
(b) Seagate Cheetah 36ES, Linear Workload.



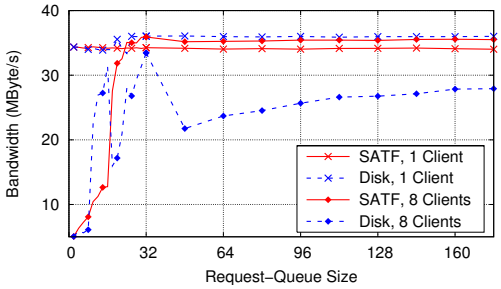
(c) IBM Ultrastar 36Z15, Pattern-nf Workload.



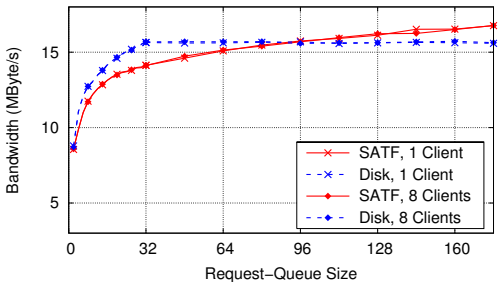
(d) Seagate Cheetah 36ES, Pattern-nf Workload.



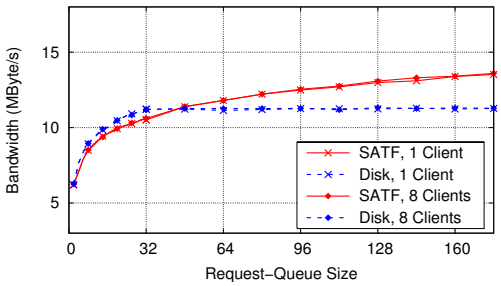
(e) IBM Ultrastar 36Z15, Pattern-hf Workload.



(f) Seagate Cheetah 36ES, Pattern-hf Workload.

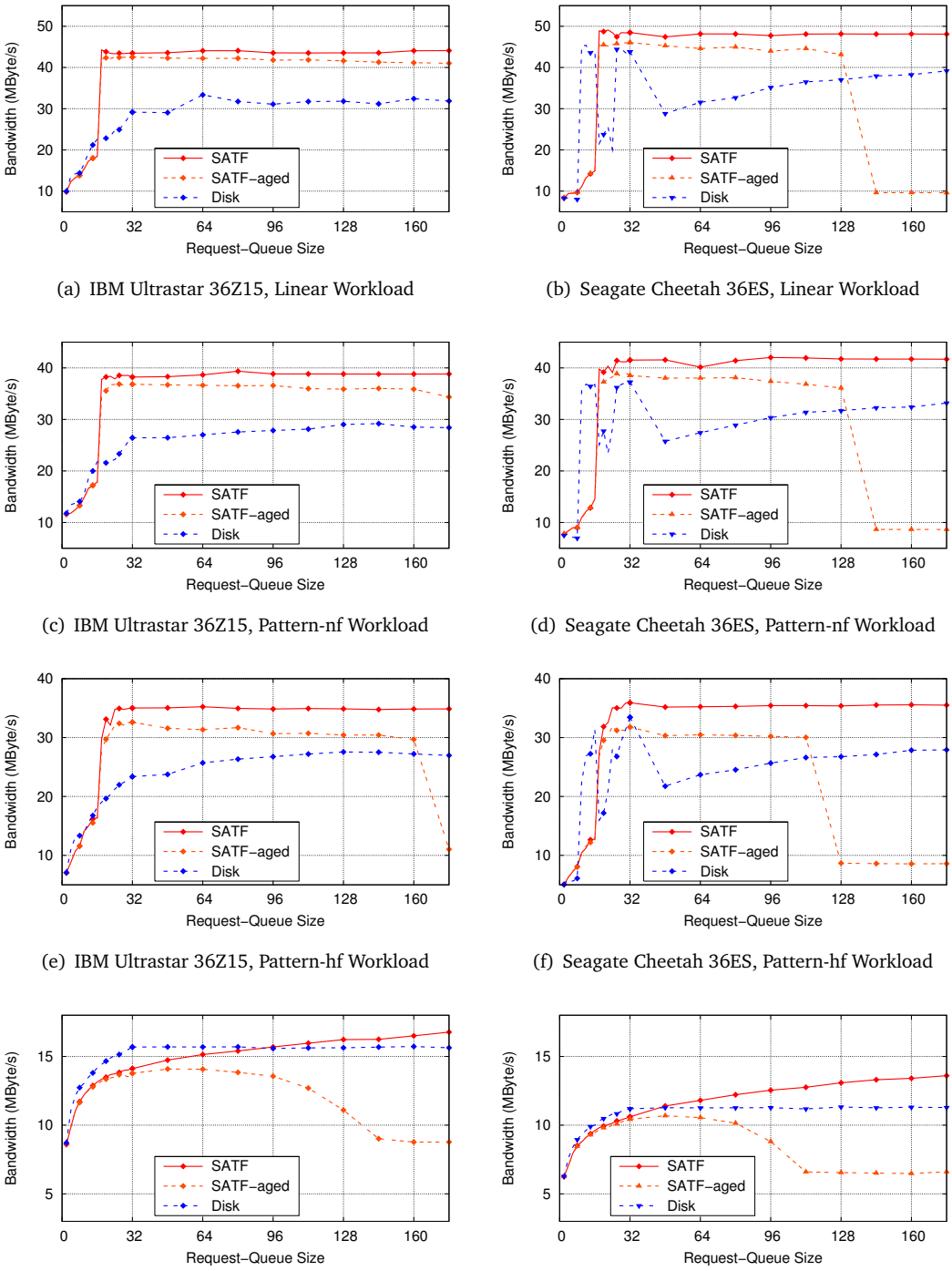


(g) IBM Ultrastar 36Z15, Random Workload.



(h) Seagate Cheetah 36ES, Random Workload.

Figure A.5: Achieved bandwidths using the SATF and the disk scheduler.



(a) IBM Ultrastar 36Z15, Linear Workload

(b) Seagate Cheetah 36ES, Linear Workload

(c) IBM Ultrastar 36Z15, Pattern-nf Workload

(d) Seagate Cheetah 36ES, Pattern-nf Workload

(e) IBM Ultrastar 36Z15, Pattern-hf Workload

(f) Seagate Cheetah 36ES, Pattern-hf Workload

(g) IBM Ultrastar 36Z15, Random Workload

(h) Seagate Cheetah 36ES, Random Workload

Figure A.6: Achieved bandwidths using the SATF scheduler, a modified SATF scheduler forcing the execution of aged requests and the disk scheduler.

References

[1] E2fsprogs Home Page. `http://e2fsprogs.sourceforge.net/`.

[2] Ext2fs Home Page. `http://e2fsprogs.sourceforge.net/ext2.html`.

[3] TIOTEST, Threaded I/O Test. `http://sourceforge.net/projects/tiobench/`.

[4] Luca Abeni and Giorgio Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proceeding of the 19th IEEE Real-Time Systems Symposium (RTSS)*, pages 4–13, Madrid, Spain, December 1998.

[5] Luca Abeni and Giorgio Buttazzo. Stochastic Analysis of a Reservation Based System. In *Proceedings of the Ninth International Workshop on Parallel and Distributed Real-Time Systems*, pages 946–952, San Francisco, USA, April 2001.

[6] Alia K. Atlas and Azer Bestavros. Statistical Rate Monotonic Scheduling. In *Proceedings of the 18th IEEE Real-Time Systems Sysmposium (RTSS)*, Madrid, Spain, December 1998.

[7] Jens Axboe. Linux Block IO—present and future. In *Proceedings of the Ottawa Linux Symposium*, pages 51–61, Ottawa, Canada, July 2004.

[8] P. R. Barham. A Fresh Approach to File System Quality of Service. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, St. Louis, USA, May 1997.

[9] Paul Ronald Barham. *Devices in a Multi-Service Operating System*. PhD thesis, University of Cambridge, 1996.

[10] W.J. Bolosky, III J.S. Barrera, R.P. Draves, R.P. Fitzgerald, G.A. Gibson, M.B. Jones, S.P. Levi, N.P. Myhrvold, and R.F. Rashid. The Tiger Video Fileserver. In *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Zushi, Japan, April 1996.

[11] Peter Bosch. *Mixed-Media File Systems*. PhD thesis, University of Twente, Netherlands, 1999.

[12] Peter Bosch and Sape J. Mullender. Real-Time Disk Scheduling in a Mixed-Media File System. In *Proceedings of the Sixth Real-Time Technology and Applications Symposium (RTAS)*, Washington D.C., USA, May 2000.

[13] Jill M. Boyce and Robert D. Gaglianello. Packet Loss Effects on MPEG Video Sent Over the Public Internet. In *Proceedings of the Sixth ACM international conference on Multimedia (ACM Multimedia)*, Bristol, UK, September 1998.

[14] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Joel Wu. Developing a Complete Integrated Real-Time System. In *Proceeding of the Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, Palma de Mallorca, Spain, July 2005.

[15] Milind M. Buddhikot, Xin Jane Chen, and Dakang Wu. Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 326–337, Austin, USA, June 1998.

[16] M.J. Carey, R. Jauhari, and M. Livny. Priority in DBMS Resource Scheduling. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB)*, pages 397–410, Amsterdam, The Netherlands, August 1989.

[17] Ed Chang and Avideh Zakhor. Variable Bit Rate MPEG Video Storage on Parallel Disk Arrays. In *Proceedings of SPIE Conference on Visual Communication and Image Processing*, pages 47–60, Chicago, USA, September 1994.

[18] Mon-Song Chen, Dilip D. Kandlur, and Philip S. Yu. Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams. In *Proceedings of the first ACM international conference on Multimedia*, pages 235–242, Anaheim, USA, August 1993.

[19] Helen Custer. *Inside the Windows NT File System*. Microsoft Press, 1994.

[20] John R. Douceur and William J. Bolosky. A Large-Scale Study of File-System Contents. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 59–70, Atlanta, USA, May 1999.

[21] Peter Druschel and Larry L. Peterson. Fbufs: A High-Bandwidth Cross-Domain Transfer Facility. In *Proceedings of the 14th ACM Symposium on Operating System Principles (SOSP)*, pages 189–202, Asheville, USA, December 1993.

[22] José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic Analysis of Periodic Real-Time Systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)*, pages 289–300, Austin, USA, December 2002.

[23] Nick Feamster and Hari Balakrishnan. Packet Loss Recovery for Streaming Video. In *Proceedings of 12th International Packet Video Workshop*, Pittsburgh, USA, April 2002.

[24] Jim Gemmell, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, and Lawrence A. Rowe. Multimedia Storage Servers: A Tutorial. *IEEE Computer*, 28(5):40–49, 1995.

[25] Shahram Ghandeharizadeh, Roger Zimmermann, Weifeng Shi, Reza Rejaie, Douglas J. Ierardi, and Ta-Wei Li. Mitra: A Scalable Continuous Media Server. *Multimedia Tools and Applications Journal*, 5(1), July 1997.

[26] Dominic Giampaolo. *Practical File System Design with the Be File System*. Morgan Kaufmann Publishers, Inc., 1999.

[27] Moving Picture Experts Group. MPEG. `http://www.chiariglione.org/mpeg/`.

[28] TU Dresden Operating Systems Research Group. L4Env—An Environment for L4 Applications. `http://os.inf.tu-dresden.de/l4env/doc/l4env-concept/l4env.pdf`, June 2003.

[29] Pal Halvorsen, Carsten Griwodz, Vera Goebel, Ketil Lund, Thomas Plagemann, and Jonathan Walpole. Storage System Support for Continuous-Media Applications, Part 1: Requirements and Single-Disk Issues. *IEEE Distributed Systems Online*, 5(1), January 2004.

[30] Claude-Joachim Hamann. On the Quantitative Specification of Jitter Constrained Periodic Streams. In *Proceedings of the Fifth IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 171–176, Haifa, Israel, January 1997.

[31] Claude-Joachim Hamann, Jork Löser, Lars Reuther, Sebastian Schönberg, Jean Wolter, and Hermann Härtig. Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, pages 119–128, London, UK, December 2001.

[32] Claude-Joachim Hamann, Andreas Märcz, and Klaus Meyer-Wegener. Buffer Optimization in Realtime Media Servers Using Jitter-Constrained Periodic Streams. Technical report, Technische Universität Dresden, 2001.

[33] Claude-Joachim Hamann and Lars Reuther. Pufferdimensionierung für schwankungsbeschränkte Ströme in DROPS (in German). In *Proceedings of 10. GI/ITG Fachtagung MMB*, Trier, Germany, September 1999.

[34] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. The Performance of Microkernel-Based Systems. In *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP)*, St. Malo, France, October 1997.

[35] Christian Helmuth. Generische Portierung von Linux-Gerätetreibern auf die DROPS-Architektur (in German). Master's thesis, TU Dresden, 2001.

[36] Michael Homuth. *Pragmatic nonblocking synchronization for real-time systems*. PhD thesis, Technische Universität Dresden, 2002.

[37] Hermann Härtig, Robert Baumgartl, Martin Borriss, Claude-Joachim Hamann, Michael Hohmuth, Frank Mehnert, Lars Reuther, Sebastin Schönberg, and Jean Wolter. DROPS - OS Support for Distributed Multimedia Applications. In *Proceedings of the Eigth ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[38] Hermann Härtig, Jork Löser, Frank Mehnert, Lars Reuther, Martin Pohlack, and Alexander Warg. An I/O Architecture for Mikrokernel-Based Operating Systems. Technical Report TUD-FI03-08, Technische Universität Dresden, July 2003.

[39] Hermann Härtig, Lars Reuther, Jean Wolter, Martin Borriss, and Torsten Paul. Cooperating Resource Managers. In *Proceedings of Workshop on QoS Support for Real-Time Internet Applications*, Vancouver, Canada, June 1999.

[40] IBM. Ultrastar 36Z15 Data Sheet. `http://www.hitachigst.com/tech/techlib.nsf/techdocs/85256AB8006A31E5872%56A78007959CE/$file/U36Z15_ds.pdf`.

[41] IEEE and The Open Group. *Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API)*.

[42] David M. Jacobson and John Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7rev1, HP Laboratories, March 1991.

[43] Kevin Jeffay, Donald F. Stanat, and Charles U. Martel. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. In *Proceedings of the 12th IEEE Real-Time Systems Symposium (RTSS)*, pages 129–139, San Antonio, USA, December 1991.

[44] Sooyong Kang and Heon Y. Yeom. Statistical Admission Control for Soft Real-Time VOD Servers. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 579–584, Como, Italy, March 2000.

[45] Philip D.L. Koch. Disk File Allocation Based on the Buddy System. *ACM Transactions on Computer Systems (TOCS)*, 5(4):352–370, November 1987.

[46] John P. Lehoczky and Sandra Ramos-Thuel. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Phoenix, USA, December 1992.

[47] Ian M. Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul T. Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1280–1297, 1996.

[48] Jochen Liedtke. On Microkernel Construction. In *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP)*, Copper Mountain Resort, USA, December 1995.

[49] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):40–61, 1973.

[50] Jane W. S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[51] Jork Löser, Lars Reuther, and Hermann Härtig. A Streaming Interface for Real-Time Interprocess Communication. Technical Report TUD-FI01-09-August-2001, Technische Universität Dresden, August 2001.

[52] C. Martin, P. Narayan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini Multimedia Storage System. *Journal of Digital Libraries*, 1997.

[53] Marshall K. Mckusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. A Fast File System for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, August 1984.

[54] Jean M. McManus and Keith W. Ross. Video-on-Demand Over ATM: Constant-Rate Transmission and Transport. *IEEE Journal on Selected Areas in Communications*, 14(6):1087–1098, August 1996.

[55] Larry W. McVoy and Steve R. Kleiman. Extent-like Performance from a UNIX File System. In *Proceedings of the USENIX Winter Technical Conference*, pages 33–43, Dallas, USA, January 1991.

[56] Anastasio Molano, Kanaka Juvva, and Raj Rajkumar. Real-Time Filesystems: Guaranteeing Timing Constraints for Disk Accesses in RT-Mach. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, San Francisco, USA, December 1997.

[57] G. Nerjes, Y. Robogianakis, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum. Scheduling Strategies for Mixed Workloads in Multimedia Information Servers. In *Proceedings of the Workshop on Research Issues in Database Engineering (RIDE)*, pages 121–128, Orlando, USA, February 1998.

[58] Guido Nerjes, Peter Muth, and Gerhard Weikum. Stochastic Service Guarantees for Continuous Data on Multi-Zone Disks. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 154–160, Tucson, USA, May 1997.

[59] John K. Ousterhout, Hervg Da Costa, David Harrison, John A. Kunzeand Mike Kupfer, and James G. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles (SOSP)*, pages 15–24, Orcas Island, USA, December 1985.

[60] Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel. IO-Lite: A Unified I/O Buffering and Caching System. *ACM Transactions on Computer Systems*, 18(1):37–66, February 2000.

[61] Thomas Plagemann, Vera Goebel, Pal Halvorsen, and Otto Anshus. Operating System Support for Multimedia Systems. *The Computer Communications Journal*, 23(3):267–289, February 2000.

[62] Martin Pohlack. Ermittlung von Festplatten-Echtzeiteigenschaften (in German). Undergraduate thesis, Technische Universität Dresden, 2002.

[63] Martin Pohlack. Plattenscheduling für Quality-Assuring-Scheduling (in German). Master's thesis, Technische Universität Dresden, 2003.

[64] A.L. Narasimha Reddy and James C. Wyllie. Disk scheduling in a multimedia I/O system. In *Proceedings of the first ACM international conference on Multimedia*, pages 225–233, Anaheim, USA, August 1993.

[65] Hans Reiser. Reiserfs. `http://www.namesys.com/`.

[66] Lars Reuther and Martin Pohlack. Rotational-Position-Aware Real-Time Disk Scheduling Using a Dynamic Active Subset (DAS). In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS)*, Cancun, Mexico, December 2003.

[67] Lars Reuther and Martin Pohlack. Using SATF Scheduling in Real-Time Systems. Work-in-Progess Report, Second USENIX Conference on File and Storage Technologies (FAST), San Francisco, USA, March 2003.

[68] Chris Ruemmler and John Wilkes. UNIX Disk Access Patterns. In *Proceedings of the Winter USENIX Conference*, San Diego, USA, January 1993.

[69] Chris Ruemmler and John Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, 27(3):17–28, March 1994.

[70] James Salehi, Zhi-Li Zhang, James F. Kurose, and Don Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements Through Optimal Smoothing. *IEEE/ACM Transactions on Networking*, 6(4):397–410, August 1998.

[71] Rainer Schlittgen. *Einführung in die Statistik*. R. Oldenbourg Verlag, München, 2003.

[72] Friedhelm Schmidt. *SCSI-Bus und IDE-Schnittstelle*. Addison-Wesley, 1994.

[73] Seagate. Cheetah 36ES Data Sheet. `http://www.seagate.com/docs/pdf/datasheet/disc/ds_cheetah36es.pdf`.

[74] Margo Seltzer, Peter Chen, and John Ousterhout. Disk Scheduling Revisited. In *Proceedings of the Winter USENIX Conference*, Washington D.C., USA, January 1990.

[75] P. J. Shenoy, P. Goyal, S. Rao, and H. M. Vin. Symphony: An Integrated Multimedia File System. In *Proceedings of ACM/SPIE Multimedia Computing and Networking (MMCN)*, pages 124–138, San Jose, USA, January 1998.

[76] P. J. Shenoy, P. Goyal, and H. M. Vin. Architectural Considerations for Next Generation File Systems. In *Proceedings of the Seventh ACM International Conference on Multimedia (ACM Multimedia)*, pages 457–467, Orlando, USA, October 1999.

[77] P. J. Shenoy and H. M. Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. In *Proccedings of SIGMETRICS / PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, Madison, USA, June 1998.

[78] Prashant Shenoy, Pawan Goyal, and Harrick M. Vin. Architectural Considerations for Next Generation File Systems. In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 457–467, Orlando, USA, October 1999.

[79] Elizabeth Shriver, Arif Merchant, and John Wilkes. An Analytic Behavior Model for Disk Drives With Readahead Caches and Request Reordering. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, Madison, USA, June 1998.

[80] Keith A. Smith and Margo I. Seltzer. File System Aging — Increasing the Relevance of File System Benchmarks. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Seattle, USA, June 1997.

[81] Keith Arnold Smith. *Workload-Specific File System Benchmarks*. PhD thesis, Harvard University, Cambridge, USA, January 2001.

[82] Udo Steinberg. Quality-Assuring Scheduling in the Fiasco Microkernel. Master's thesis, Technische Universität Dresden, 2004.

[83] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the XFS File System. In *Proceedings of the USENIX Technical Conference*, pages 1–14, San Diego, USA, January 1996.

[84] Adobe Systems. Understanding and Using High-Definition Video. White Paper, 2004.

[85] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, Inc., 2001.

[86] Tekram. User Manual for DC-390U3 Series. `http://www.tekram.com/downloads/Storage/SCSI/PCI/DC-390X/DC-390U3D/Manu%al/DC-390U3_SERIES.ZIP`.

[87] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times. In *Proceedings of the Real-Time Technology and Applications Symposium (RTAS)*, pages 164–173, Chicago, USA, May 1995.

[88] Frank Unger. Pufferdimensionierung für Schwankungsbeschränkte Ströme (in German). Undergraduate thesis, Technische Universität Dresden, 1998.

[89] H. M. Vin, P. Goyal, A. Goyal, and A. Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of the Second ACM International Conference on Multimedia (ACM Multimedia)*, pages 33–40, San Francisco, USA, October 1994.

[90] Harrick M. Vin, Alok Goyal, Anshuman Goyal, and Pawan Goyal. An Observation-Based Admission Control Algorithm for Multimedia Servers. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS)*, pages 234–243, Boston, USA, May 1994.

[91] Ravi Wijayaratne and A. L. Narasimha Reddy. Integrated QOS management for disk I/O. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pages 487–492, Los Alamitos, USA, June 1999.

[92] Ravi Wijayaratne and A. L. Narasimha Reddy. Providing QOS Guarantees for Disk I/O. *Multimedia Systems*, 8(1):57–68, January 2000.

[93] Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles. Dynamic Storage Allocation: A Survey and Critical Review. In *Proceeding of the International Workshop on Memory Management*, Kinross, UK, September 1995.

[94] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt, and John Wilkes. On-Line Extraction of SCSI Disk Drive Parameters. Technical Report CSE-TR-323-96, University of Michigan, 1996.

[95] Joel C. Wu and Scott A. Brandt. Storage Access Support for Soft Real-Time Applications. In *Proceedings of the 10^{th} IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Toronto, Canada, May 2004.

[96] Philip S. Yu, Ming-Syan Chen, and Dilip D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 44–55, La Jolla, USA, November 1992.

[97] Roger Zimmermann and Kun Fu. Comprehensive statistical admission control for streaming media servers. In *Proceedings of the 11^{th} ACM International Multimedia Conference (ACM Multimedia)*, pages 75–85, Berkeley, USA, November 2003.

[98] B. Özden, R. Rastogi, and A. Silberschatz. Disk Striping in Video Server Environments. In *Proceedings IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, Hiroshima, Japan, June 1996.